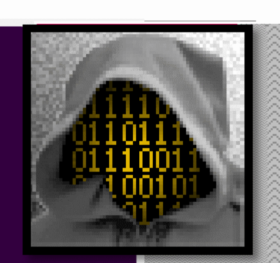
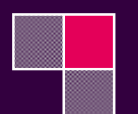


تراكيب البيانات والخوارزميات

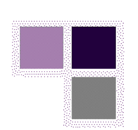
د. حسين صالح ابوصره

د: علوم حاسب - هندسة حاسب



D.Hesin Abo-Soorah
www.sarah4all.worldpress.com
++C

المحتويات



بسم الله الرحمن الرحيم

مقدمة

يُعد مفهوما البيانات Date والمعلومات Information من المفاهيم الأساسية في دراسة الحاسبات واستخدامها ، ومن هنا يجب أن نحدد المقصود بمعنى كلاً منهما..

- المعلومات : هي كل البيانات والمعاني التي تحملها تلك البيانات وليس البيانات فقط ، فمثلاً الكتاب عندما يكون كتابي أنا لا يكون بالنسبة لي حقيقة مجردة (بيانية) وحسب وإنما يكون كتابي الذي أعرفه وقد اشتريته في تاريخ معين وقد قرأته ، أي بمعنى أن الكتاب أرتبط بي بحقائق معينة.

- البيانات : يمكن أن نقصد بالبيانات أنها الحقائق والمفاهيم الموضوعية التي نراها ونتعامل معها في حياتنا اليومية أو بمعنى آخر فإن البيانات هي جميع الأشياء والحقائق التي لا نشعر بوجود علاقة أو معرفة مباشرة أو غير مباشرة بها .

فمثلاً: كتاب،سيارة،جامعة.......الخ ، هي كلمات لها مدلولات خاصة بها ومستقلة عنّا إذن فهي حقائق موضوعية نراها ونستخدمها كما هي.

#كما ويمكن تعريف البيانات على أنها مجموعة من القيم وظيفتها التعبير عن الكينونات أو الأحداث التي تعبر عنها وبالتالي فإنها مختلفة فيما بينها من حيث نوع العمليات التي تجري عليها حسابية أو منطقية أو غير ذلك.

- بناءً عليه فإننا نقول....

١- أن لدينا بيانات عددية عندما نتعامل مع الأرقام .

٢- بيانات رمزية عندما نتعامل مع الحروف والكلمات .

٣- بيانات منطقية عندما نتعامل مع بيانات تحتمل إجابات الصواب أو الخطأ .

- **وهذه الأنواع المذكورة أعلاه تسمى باسم أنماط بيانية بسيطة (Simple date types) .**

- وبالمقابل هناك العديد من الحقائق المختلفة التي يصعب متابعتها والإحاطة بها وبجميع جزئياتها أحيانا وهذا لتعدد البيانات الذي ينطوي عليه وصف هذه الحقائق مما يدفعنا إلى تنظيم هذه البيانات في مجموعات ، جداول ، قوائم أو سجلات أو غير ذلك من أساليب التنظيم ويُطلق على هذا النوع من البيانات باسم أنماط بيانية مركبة (Structured date types).

- يمكن تلخيص ما ذكر أعلاه في الجدول التالي...

أنماط بسيطة		أنماط مركبة
Boolean **Character** **Integer** **Real**		Record Set Array String File

وعلى هذا الأساس فإن المصطلح تركيب بياني Date structure يشير بشكل مباشر إلى مجموعة من عناصر البيانات التي لها تنظيم خاص وكذلك لها أسلوب في الوصول إلى العناصر الفردية سواءً من خلال عملية تخزين القيم أو خلال عملية استرجاعها ، وبناءً على هذا المفهوم فإن هناك ميزتين أساسيتين تتصف بهما تراكيب البيانات وهما :

١- يمكن التعامل مع كل عنصر منهما على حدة ، كما لو كان متغيراً مستقلاً.

٢- طريقة تنظيم العناصر تؤثر مباشرةً على طريقة الوصول إلى العنصر الواحد أو التعامل مع المركب البياني ككل.

- **أما من جهةٍ أخرى فإن مصطلح تركيب البيانات فإنه يُشير إلى الطرق والأساليب المختلفة التي يمكن من خلالها ترجمة التصور المنطقي للبيانات(كما يراها المبرمج) أي أنه يهتم بالطرق المختلفة لتنظيم البيانات وبالخوارزميات اللازمة لمعالجة هذه البيانات في ذاكرة الحاسوب.**

← ومن هنا ينبثق تعريف الخوارزمية (Algorithm) :-

أنها مجموعة الخطوات اللازمة لحل مسألة معينة ويجب أن تتوفر بها الشروط التالية..

١- أن تكون مختصرة الخطوات وواضحة وبسيطة ودقيقة.

٢- أن تكون قابلة للتنفيذ.

٣- أن يكون لها نهاية تتوقف عندها المعالجة Execution .

← وتكون الخوارزمية جيدة إذا كانت تتصف بما يلي...

١- مكتوبة بأسلوب واضح.

٢- سهلة البرمجة.

٣- سريعة التنفيذ.

٤- عند تطبيقها على الحاسبات تكون رخيصة الثمن.

التراكيب التجريدية

(Abstract date types)

←التراكيب التجريدية هي: تراكيب البيانات التي تستطيع استخدامها من خلال عمليات معينة على شكل إجراءات و اقترانات على هذه التراكيب من دون الحاجة الى معرفة تفاصيل تمثيل هذه التراكيب.

- على سبيل المثال ي لغة ++C ⟵ Integer date type هي نوع من أنواع البيانات التجريدية بالنسبة لنا إذ أننا نستخدم هذا النوع من خلال عمليات معرّفة عليها مثل : الضرب ، الجمع ، القسمة الطرح وبدون أن نضطر إلى معرفة كيفية تمثيل الأعداد الصحيحة فعلياً .

- **بمعنى آخر .. عند تصميم وإنشاء تركيبة بيانات جديدة فإنه يجب إنشائها كتركيبة بيانات تجريدية بحيث نستطيع (أو يستطيع آخرون) من استخدام هذه التركيبة فيما بعد من خلال عمليات معرّفة عليها دون الحاجة إلى معرفة تفاصيل كيفية تمثيلها وتنفيذها . (أي أننا قمنا بإخفاء كيفية تمثيل للمعلومات - Information hiding).**

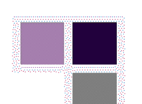
←فعلى سبيل المثال : لو أردنا إنشاء تركيبة بيانات لتمثيل قائمة من المعلومات الخاصة بطلبة ولنطلق عليها اسم Student List فيجب تصميمها وإنشائها بحيث نستطيع نحن أو الآخرون من استخدامها دون الحاجة إلى معرفة تفاصيل تمثيل هذه القائمة ، وأفضل أسلوب أن نمثلها على شكل مصفوفة من السجلات وهي تعرف باسم (array of records) وعندها كي نستطيع استخدام تركيبة البيانات هذه هو معرفة العمليات (Operations) المُعرفة عليها وكيفية استدعاء هذه العمليات ، ففي مثالنا مجموعة العمليات المطلوبة هي :

١- إدخال طالب جديد للقائمة List insert .

٢- عملية البحث عن طالب معين في القائمة List search .

٣- حذف طالب من القائمة List delete .

٤- طباعة القائمة List display .

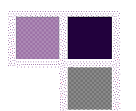


← وعادةً ما تنفذ هذه العمليات كإجراءات Procedures و اقترانات Functions .

بناءً على ما سبق يمكن استخلاص فائدة استخدام التراكيب التجريدية بأن المبرمج الذي يحاول استخدام تركيبة البيانات هذه في برامج تطبيقية كبيرة ومعقدة عندها فهو يركز تفكيره على المسألة التي يحاول برمجتها دون بذل جهد في تفاصيل تمثيل تركيبة البيانات مما يجعل البرنامج أكثر فاعلية وأقل تعقيداً ، وكذلك سهل الكتابة والفهم والتعديل .

فوائد هيكل البيانات :-

١. اختصار زمن التخزين واسترجاع البيانات
٢. تعطي المبرمج مرونة كتابة البرامج
٣. تكوين برامج متماسكة البناء والتركيب مع تسلسل منطقي
٤. توزيع صحيح للبيانات في ذاكرة الحاسب



تحليل الخوارزميات Algorithms complexity

إن وقت تنفيذ الخوارزمية يعتمد على خطوات الخوارزمية لذلك فان أول خطوة في تحليل الخوارزمية هي التعبير عن عدد الخطوات في الخوارزمية بواسطة اقتران يرمز له T بدلاله حجم البيانات N أي T(n)

أ/ اذا كانت الخوارزمية تتكون من خطوة برمجيه .. على سبيل المثال ...

T (n) = 1 <= int a=4;

على سبيل المثال اذا كانت الخوارزمية مكونه من ثلاث خطوات ..

```
Int a=4;
int b=2;        T (n) = 3 <=
C=a+b;
```

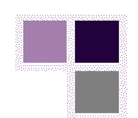
نلاحظ ان هذه الخوارزمية تتكون من خطوة واحدة قبل جملة الدوران وجملتين داخل الدوران ..

```
Sum = 0;
For ( i=1;i<=n;i++){
sum=sum+A[i];
cout<<sum;
}
```

وعليه فان الخطوة الأولى خارج الدوران T(n) لها T(n)=1

أما الجملتين داخل الدوران فان كل جمله تتكرر n مرة وعليه .. T(n)=n+n

وعليه فان لكل الخوارزمية ← T(n)=1+2n أي عدد الخطوات اللازمة لتنفيذ الخوارزمية كاملة ..



- وقد لجأ محللو الخوارزميات الى طريقة رمزية notation تدعى ترميز O الكبيرة big_ O notation
- لأستخلاص العوامل المهمة في التعبير عن كفاءة الخوارزمية

ولذلك لنفترض أن هناك أقتراناً $f(n)$ معرفاً على الأرقام غير السالبة بحيث يكون :

$T(n) \le c*f(n)$ عدد الخطوات اللازمة لتنفيذ الخوارزمية

Where $n \ge m$

حيث ان c,m ثابتين عدديان ..

- أي أنه إذا كان عدد خطوات الخوارزمية $T(n)$ أقل أو يساوي حاصل ضرب الثابت c في $f(n)$ عندما تكون n قيمة كبيرة أكبر من الثابت m عندها نستطيع القول بان الخوارزمية هي $O(f(n))$ وعندها يدعى الثابت c بثابت التناسب constant of proportionality
- لنعود للمثال السابق (مثال الدوران) ونحاول ايجاد $f(n)$

المطلوب :

كما وجدنا أن:

$T(n)=1+2n \le 2n$

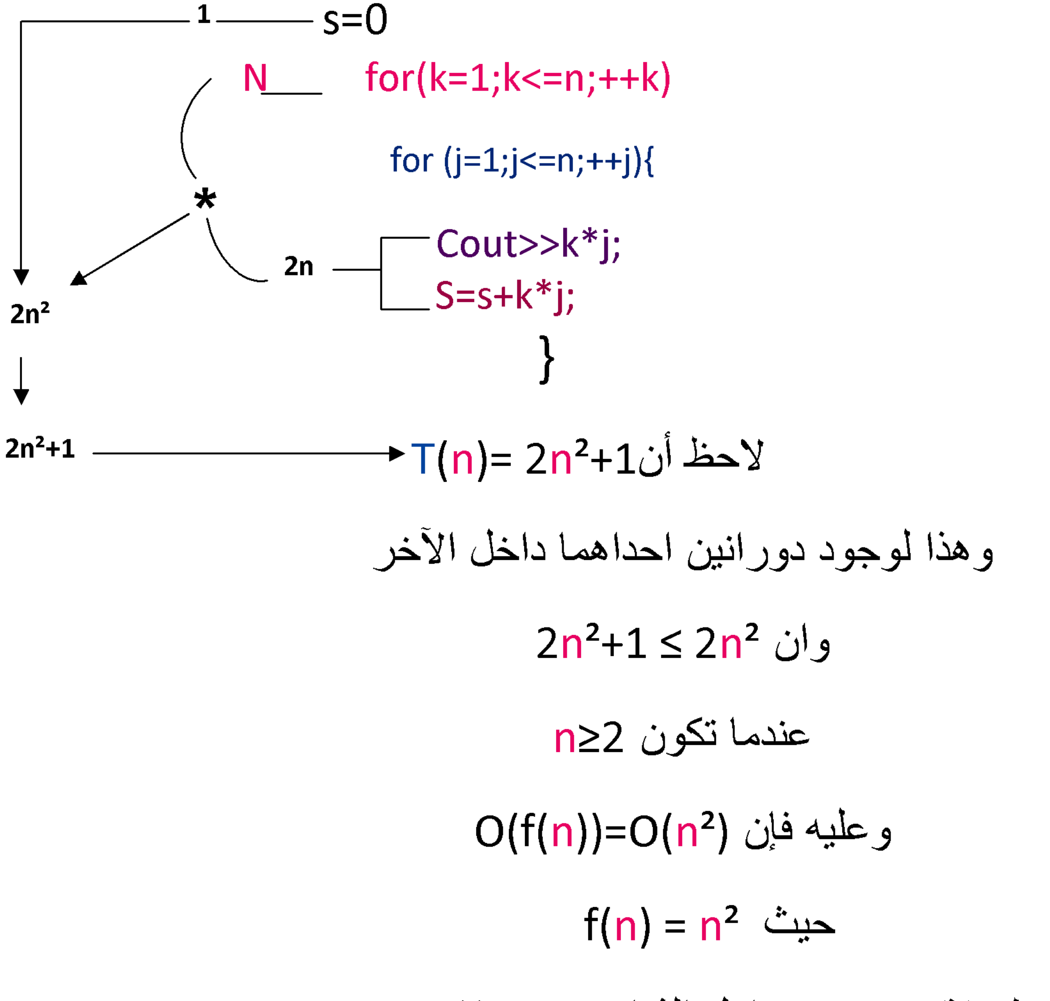
$n \ge 2$

حيث ان ثابت التناسب $c=2$ وان قيمة $m=2$ وان $f(n)=n$

وعليه نقول ان الخوارزمية هي .. $O(f(n)) \rightarrow O(n)$

مثال ..

Find the complexity for the following Algorithm



لاحظ أن $T(n)= 2n^2+1$

وهذا لوجود دورانين احداهما داخل الآخر

وان $2n^2+1 \leq 2n^2$

عندما تكون $n \geq 2$

وعليه فإن $O(f(n))=O(n^2)$

حيث $f(n) = n^2$

* نلاحظ أن طريقة big-O تتجاهل الثوابت وهي لا تفرق بين خوارزميات عدد خطواتها n^2 او $2n^2$ او حتى $100n^2$ وتعتبرهم جميعا $O(n^2)$

☺ ملاحظة :-

غالبا ما يعتمد عدد تنفيذ خطوات خوارزمية معينه على طبيعة البيانات، لذلك عادة ما نحلل الخوارزمية أخذين ثلاث معاملات أخذين بعين الاعتبار :

١/ الحالة الأفضل Best case :- وهي الحالة التي تحتاج الى تنفيذ أقل عدد من الخطوات.

٢/ الحالة الأسوأ worst case :- وهي الحالة التي تحتاج فيها الخوارزمية الى تنفيذ أكبر عدد من الخطوات.

٣/ الحالة الوسطى average case :- وهي الحالة التي تتطلب تنفيذ معتدل من الخطوات.

على سبيل المثال البحث الخطي Linear Search التي تبحث عن قيمة معينة X في مصفوفة A مكونة من n عنصر .

```
i=1,
found=false;
while i<=n do
if A[i]=x Then
found=false;
else
i=i+1;
```

١ / الحالة الأفضل Best case عندها قد تنفذ عمليه مقارنه واحده ، وذلك إذا كانت القيمة التي نبحث عنها مساوية للقيمة المخزنة في العنصر الأول وفي هذه الحالة تكون الخوارزمية O(1)

٢/ Worst case :- اذا احتاجت الخوارزمية الى n من عمليات المقارنة اذا كانت القيمة التي نبحث عنها هي العنصر الأخير في المصفوفة او لم تكن موجودة في المصفوفة وعندها تكون الخوارزمية O(n)

٣/ Average case :- ان تحليل الحالة الوسطى هي العملية الأعقد وذلك لكونها تعتمد على معلومات احصائية عن احتمال وجود القيمة في الموقع الأول في المصفوفة أو الثاني .. أو الأخير .. وعليه اذا افترضنا توزيعا متكافئا uniform distribution للبيانات فان احتمال وجود القيمة المبتغاه هو p=1\n وبالتالي زمن تنفيذ الخوارزمية يحسب بالعلاقة التالية

C(n)= (1*1\2)+(2*1\2)+…+(n*1\n)=(n+0)\2

وعليه تكون الحالة المتوسط Av.case أيضا O(n)

- ملاحظة ..

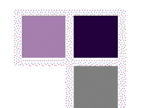
غالبا ما نركز على تحليل Best and worst case ونعمل الـ

Average case لصعوبة تحليلها بطريقة دقيقة .

عادة الاقتران المقياسي complexity ⟵ f(n) يقارن مع بعض المقاييس الزمنية الكثر انتشارا

$O(1)<O(\log n)<O(n)<O(n \log n)<O(n^2)<O(n^3)$

وهي مرتبه حسب قوتها من اليسار لليمين .



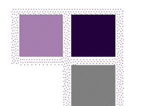
أ/ الزمن الثابت constant computing time
ويشار له O(1) أي ان الوقت المطلوب للعملية لا يتغير.
وهذا يظهر في عمليات Read, write كذلك عند الحاجه الى الوصول الى احد عناصر مصفوفه واستبدال قيمة بقيمة اخرى في عمليات الفرز.

ب/ الزمن الخطي Liner time
ويشار له O(n) وهي خوارزمية ذات زمن يتناسب طرديا مع حجم المشكلة أي مع زيادة حجم البيانات فان الزمن يتزايد.
فمثلا في عمليات إيجاد أحد العناصر في قائمة متصلة Linked list

ج/ الزمن اللوغاريتمي Logarithmic time
ويشار له O(logn) وهنا الخوارزمية تعمل اكثر من الخوارزمية ذات الوقت الثابت وبالتالي تحتاج الى وقت اطول منها ولكنها اقل عن الوقت الذي تستغرقه خوارزمية ذات وقت خطي .

وتستخدم هذه الحالة عند استخدام Binary search tree
* في حالة احتاجت الخوارزمية الى زمن بشكل يفوق الزمينين الخطي واللوغاريتمي فاننا نستخدم O(n log n) ومن الحالات المستخدمة هنا عملية الـ merge sort

د/ الزمن المتعدد الحدود polynomial time
وهذا عندما تكون العلاقة الزمنية محكومة بمتغير متعدد الحدود فان نمو العلاقة يزداد بمعدل تربيعي quadratic $O(n^2)$ او
cubic $O(n^3)$ او $O(n^a)$
وعليه فان الخوارزمية ذات الكفاءة$O(n^2)$ تقوم بأعمال أكثر مما تقوم به خوارزمية ذات وقت خطي وكذلك $O(n^3)$ تستغرق وقتا أطول من $O(n^2)$
على سبيل المثال تخزين قيمة ابتدائية في كل عنصر من عناصر مصفوفة ذات ثلاث أبعاد مما يستغرق زمنا $O(n^3)$



1\ The Bubble Sort Algorithm

```
for(i=1 ; i<=n-1 ; ++i)

For(j=n ; i=1 ; --i){

If a [j-1] > A[j] then

Temp = A[j-1];

A[j-1]=A[j];

A[j]=temp;

}
```

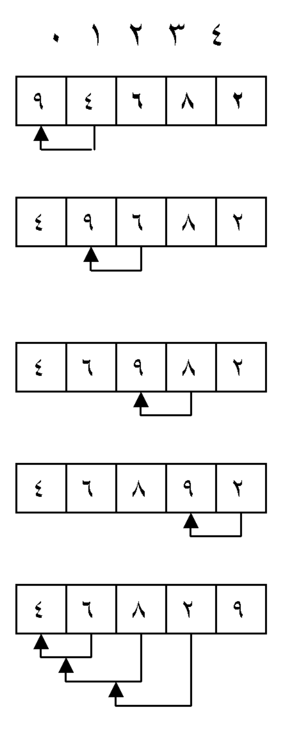
نلاحظ أن الخطوة الحرجة The critical operation (وهي العملية الأكثر تكراراً في الخوارزمية) هي عملية المقارنة if وعدد تكراراها هو المجموع ..

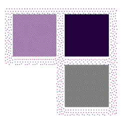
T(n) = (n-1) + (n-2) + (n-3) + ... +3+2=

[((n-1)+2)(n-2)]\2=

[(n+1)(n-2)] \2 = (0.5 n^2)-(0.5n)-1

وبهذا يتضح أن الخوارزمية $O(n^2)$





2\ The selection Sort Algorithm..

```
For(i=1 ; i<=n-1 ; ++i)
{find the smallest element in the unsorted part}
Min=X[i]
Position = i
For(J=i+1;j<=N;j++)
If X[J]<min then
Min=X[j]
Position =J
End; {if}

{swap the smallest element (x [position with x[i]}
Temp = x[i]
X[i]=x[position]
x[position]=Temp
end{for}
```

* العمليات الحرجة هي جملة المقارنة if والموجودة داخل دوارين متداخلين nested loop وعدد تكرار جملة المقارنة if هو المجموع ..

$T(n)=(N-1)+(N-2)+(N-3)+(N-4)+ \ldots + 3+2+1 =$

$[N* (n-1)]\backslash 2 =(0.5* N^2)-(0.5*N)$

وعليه فإن $O(n^2)$

1\ complexity of insertion sort

Worst case:-

$T(n)=1+2+\ldots+(n-1)= [n(n-1)]\backslash 2=O(n^2)$

Average case:-

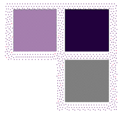
$T(n)= (1\backslash 2)+(2\backslash 2)+\ldots.+(n-1)\backslash 2 = [n(n-1)]\backslash 4=O(n^2)$

2\ Marge-sort

$T(n)=n \log n \rightarrow O(n \log n)$

افرض أن T(n) والتي تقوم بفرز n عنصر في المتتالية A فاننا نحتاج n log مرحلة passes بحيث انه في كل pass يدمج n عنصر ..

وعليه فان O(n log n)




Arrays


المصفوفات

تعرف الـ array على أنها مجموعة من العناصر المتشابهة ويمكن للعناصر أن تكون قيمة رقمية أو رمزية أو خليط منها أو مجموعة من السجلات .

وتصنف المتجهات إلى صنفين من حيث الأبعاد :

١- One-dimensional array (ذات بعد واحد) .

٢- متجهات متعددة الأبعاد matrix .



❖ المصفوفات الأحادية: (one-dimensional array)

وفي بعض الأحيان يطلق عليها اسم Linear array .

Let data be a 6-element linear array of integers such that:-

Data[1]=247 data[2]=56 data[3]=429

data[4]=135 data[5]=87 data[6]=156

يمكن كتابة هذا المتجه بطريقتين :-

١- Horizontal vectorوهي الأكثر انتشاراً.

Data = 247 56 429 135 87 156

٢- Vertical vector

$$Data=\begin{pmatrix}247\\56\\429\\135\\87\\156\end{pmatrix}$$

ولتوضيح مفهوم المتجه على المستوى المنطقي والمستوى التخزيني

المستوى التخزيني للهيكل البياني	
الاسم البياني	رقم خلية الذاكرة العنوان المطلق
. . . .	
A	202
B	203
MARK	204
X	209

0		
1		
2		
.	.	
.	.	
.	.	
202	20.15	A
203	14.7	B
204	65	MARK(1)
205	17	MARK(2)
206	59	MARK(3)
207	41	MARK(3)
208	89	MARK(4)
209	77.3	MARK(5)

المعادلة العامة التي يتحدد بواسطتها عنوان أي عنصر في الذاكرة لمصفوفة أحادية في مثالنا .

أي : the address of any element of array

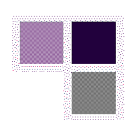
LOC (MARK (I)) = LOC (MARK) + I – 1

Where

The name of array: - **MARK**

The absolute address of array: - **LOC (MARK)**

The index of array: - **I**



مثال لحساب عنوان mark(5) كما يلي :-

LOC (MARK (5)) =204+5-1 = 208

LOC (MARK (2)) =204+2-1 = 205

المصفوفات الثنائية : (tow-dimensional array)

لنفرض أن لدينا قائمة من أربع طلاب وجدول علاماتهم لثلاثة مواضيع دراسية : رياضيات ، فيزياء ، علوم .

اسم الطالب	رياضيات	فيزياء	علوم
محمد	85	43	72
محمود	63	55	22
أحمد	93	82	47
مصطفى	50	40	77

نطلق على علامات اسم M ونكون مصفوفة من أربع سطور rows وثلاثة أعمدة

	Math 1	Phs 2	Since 3
محمد	M[1,1] 85	M[1,2] 43	M[1,3] 72
محمود	M[2,1] 22	M[2,2] 55	M[2,3] 63
أحمد	M[3,1] 47	M[3,2] 82	M[3,3] 93
مصطفى	M[4,1] 77	M[4,2] 40	M[4,3] 50

حيث نعلن المصفوفة كما يلي : m [I , j]

حيث : i index of row

:j index of columns

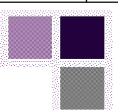
لغات البرمجة عادة تسير وفقا لأحد الأسلوبين :

التمثيل الطولي (العمودي) : column – major ordering

M[1,1]	[2,1]	[3,1]	[4,1]	[1,2]	[2,2]	[3,2]	[4,2]	[1,3]	[2,3]	[3,3]	[4,3]

التمثيل الأفقي : row – major ordering

M[1,1]	[1,2]	[1,3]	[2,1]	[2,2]	[2,3]	[3,1]	[3,2]	[3,3]	[4,1]	[4,2]	[4,3]



لنفرض أن أبعاد مصفوفة ثنائية الأبعاد والمتكونة من N من السطور و M من الأعمدة مكتوبة A(i,j) أو مصفوفة A[N*M] فإن موقع العنصر A(i,j) في الذاكرة بطريقة الأعمدة حسب المعادلة التالية :-

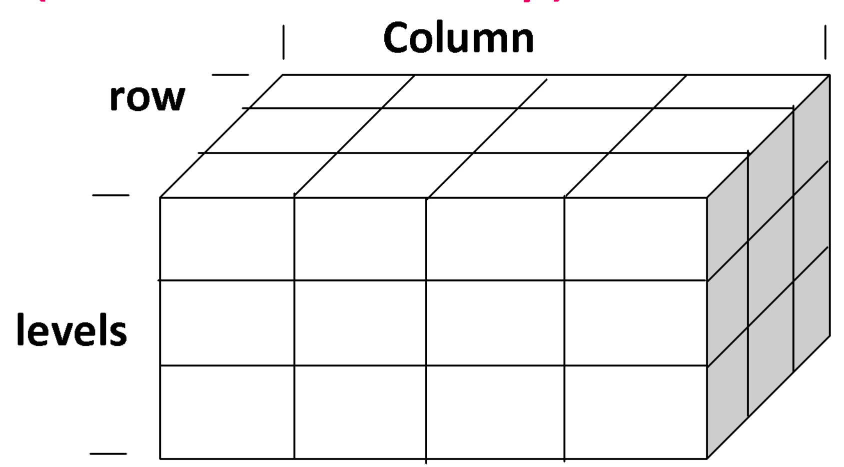
$$LOC[A(i,j)] = LOC(A) + (J-1)N + I - 1$$

حيث LOC(A) موقع أول عنصر من المصفوفة A .

ويكون موقع العنصر A(i,j) في الذاكرة حسب بطريقة السطور حسب المعادلة التالية :-

$$LOC[A(i,j)] = LOC(A) + (I-1)M + J - 1$$

❖ المصفوفات الثلاثية (three-dimensional array)



هذه مصفوفة

M (i, j, k)

i: - دليل خاص بترقيم السطور

j: - دليل خاص بترقيم الأعمدة

k: - دليل خاص لترقيم المستويات

لنفرض M(3,3,3) :- ثلاث سطور وثلاث أعمدة وثلاث مستويات

M(1,1,1)	M(1,2,1)	M (1,3,1)
M(2,1,1)	M(2,2,1)	M(2,3,1)
M(3,1,1)	M(3,2,1)	M(3,3,1)

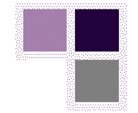
M(1,1,2)	M(1,2,2)	M(1,3,2)
M(2,1,3)	M(2,2,2)	M(2,3,1)
M(3,1,3)	M(3,2,3)	M(3,3,2)

M(1,1,2)	M(1,2,2)	M(1,3,2)
M(1,1,2)	M(1,2,2)	M(1,3,2)
M(1,1,2)	M(1,2,2)	M(1,3,2)

لتحديد موقع عنصر ما في طريقة التخزين العمودية نستخدم المعادلة التالية :

MARK [I, L, K] =LOC (MARK) + (K-1).N.M+ (J-1).M+J-1

حيث LOC(MARK) :- موقع أول خلية من المصفوفة MARK



- **تمرين : ما هي معادلة إيجاد موقع تخزين عنصر في مصفوفة ثلاثية مخزنة بالطريقة الأفقية ؟؟؟**

		1	2
		3	4
	PaG2	5	6
	7	8	
	9	10	
PaG1	11	12	
13	14		
15	16		
17	18		

التخزين بالتمثيل الطولي (بالأعمدة) :

أخذنا أولا أعمدة الصفحة الأولى ثم أخذنا أعمدة الصفحة الثانية ثم أعمدة الصفحة الثالثة

13 15 17, 14 16 18, 7 9 11, 8 10 12, 1 3 5, 2 4 6

التخزين بالتمثيل الأفقي (بالأسطر) :

حيث نخزن أول قيمة في الصف الأول من الصفحة الأولى ثم قيمة الأولى من الصف الأول في الصفحة الثانية ثم القيمة الأولى من الصف الأول من الصفحة الثالثة ... الخ

13 7 1, 14 8 2, 15 9 3, 16 10 4, 17 11 5, 18 12 6

لحجز مواقع لمتتالة Array :

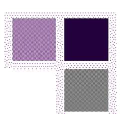
```
int A[5]
float B[4]
char [3]
```

لإدخال عناصر متتالة :

```
int A[5];
for (i=0 ; I<5 ; I++)
      cin>>A[i];
```

Sequential search in array

```
int key; //key is value you want to check if it in array
cin>>key; //to give key any value
Boolean found = false;
For (i=0; i<5; i++)
{       if( array[i] ==key) //check if value is found
            {
                  Found = true;
                  Break;
            }
}
```



😃bubble sorting😃

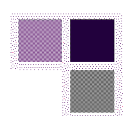
```
#include<iostream.h>
main ()
{
      int A [5] = {9, 4, 6, 8, 2};
      int temp;
      for (int i=0; i<=4; i++)
      cout<<A[i] <<" ";
      cout<<endl;
      For (i=1 ; i<=4 ; i++)
         for (j=1; j<=5-i; ++j)
           {  if ( A[j-1]>A[j] );
                {Temp= A [j-1]; //swap the value
                  A [j-1] =A[j];
                  A[j] =temp;
                }
           } //end for
      for (i=0; i<5; i++) //to print the array after sorting
      cout<<A[i] <<" ";
} //end of main
```

لحجز موقع matrix :

```
int A [5] [3]
float B [5] [3]
char C [5] [3]
```

قرأة أو إدخال مصفوفة ذات بعدين :

```
for (i=0; i<=5; i++) {
      for (j=0; j<3; j++)
            cin>>A[i] [j];
}
```



Sorting & search

الفرز sorting

تعرف عملية فرز البيانات على انها ترتيب البيانات بشكل تصاعدي او تنازلي اعتمادا على علاقه خطيه تربط عناصر هذه البيانات معا.

ولعملية الفرز عدة فوائد منها :-

١- الإقلال من حركة البيانات ومبادلتها حيث إن عملية المبادلة تأخذ جزا كبيرا وقت المعالجة.
٢- الإقلال من عملية الإدخال والإخراج وذلك بالتمكن من نقل البيانات المفروزة في وحدات فيزيائية كبيرة (blocks).
٣- توفير الوقت أثناء عملية البحث عن عنصر معين في البيانات.

- هناك عدة طرق تستخدم في عملية الفرز وتختلف هذه الطرق عن بعضها البعض في أمور عده ويمكن دراسة هذه الطرق من خلال ما توفره من :
  ١. الوقت اللازم للبرمجة .
  ٢. وقت تنفيذ البرنامج .
  ٣. الحجم اللازم من الذاكرة .

- يمكن تقسيم الفرز الى نوعين رئيسين هما :-

١- الفرز الداخلي internal sorting methods :
يستخدم هذا النوع من الفرز إذا أمكن استيعاب جميع سجلات الملف في ذاكرة الحاسوب الداخلية (الرئيسية) وبذلك يمكن تنفيذ البرنامج على جميع السجلات ، وهذا النوع من الفرز يستخدم عندما يكون عدد السجلات معقولا ويمكن استيعابه بواسطة ذاكرة الحاسوب الرئيسية في الوقت نفسه .

٢- الفرز الخارجي external sort :

ويستخدم في حالة وجود عدد كبير جدا من السجلات في الملف المراد فرز هو بالتالي لا يمكن لذاكرة الحاسوب من استيعاب جميع سجلات الملف في الوقت نفسه . لذلك يتم الاستعانة بوسائط التخزين المساعدة ، مثل الأقراص ... الخ .

١. الفرز الادخالي insertion sort :

افرض المصفوفة A متكونة من n عنصر A[1] , A[2] , … , A[N] مخزنة في الذاكرة.

فان الفرز الادخالي يمسح A من A[1] إلى A[N] مدخلا كل عنصر A[K] في موقعها الملائم في الـ subarray A[1],A[2],…,A[K-1] والمخزنة سابقا .

وعليه

pass1. A[1] by itself is trivially sorted

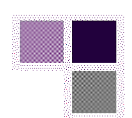
Pass2. A[2] is inserted either before or after A[1] so
That : A[1],A[2], that is sorted

Pass3. A[3] is inserted into its proper place in
A[1], A[2], that is, before A[1], between
A[1] and A[2], or after A[2], so that :
A[1], A[2], A[3] is sorted .

Pass4. A[4] is inserted into its proper place in
A[1], A[2], A[3] so that:
A [1], A [2], A [3], A[4] is sorted .

passN. A [N] is inserted into its proper place in
A [1], A [2], …, A[N-1] so that :
A[1], A[2], A[3], A[4] is sorted .

The algorithm is simplified if there is an element A[j] such that A[J] ≤ A[K] ; otherwise we must constantly check to see if we are comparing A[K] with A[1]. This condition can be accomplished by introducing a sentinel element A[0] = -∞ (or a very small number) .



pass	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K=1	-∞	77	33	44	11	88	22	66	55
K=2	-∞	77	33	44	11	88	22	66	55
K=3	-∞	33	77	44	11	88	22	66	55
K=4	-∞	33	44	77	11	88	22	66	55
K=5	-∞	11	33	44	77	88	22	66	55
K=6	-∞	11	33	44	77	88	22	66	55
K=7	-∞	11	22	33	44	77	88	66	55
K=8	-∞	11	22	33	44	66	77	88	55
	-∞	11	22	33	44	55	66	77	88

Algorithm : (insertion sort) INSERTION (A,N) this is algorithm sorts the array A with N elements

1 – set A[0] = -∞ [initializes sentinel elements] .

2 – replat steps 3 to 5 for k=2,3,…,N .

3 – set TEMP=A[K] and PTR=k-1 .

4 – repeat while TEMP<A[PTR] .

A . set A[PTR+1]=A[PTR] moves element forward

B . set PTR=PTR-1

[End of loop]

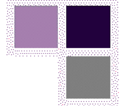
5 – set A[PTR+1]= TEMP insert element in proper place

[end of step 2 loop]

6 – return

```
INSERTION (A,N)
  A[0]=-∞;
FOR(K=2 ; K<N ;++K){
   TEMP =A[K]; PTR= k-1;
   For(;TEMP<A[K] ;){

           A[PTR+1]=A[PTR];
           PTR=PTR-1; }
          A[PTR+1]=TEMP ;}
     Return ;
```



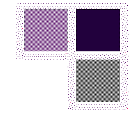
٢- الفرز الاختياري selection sort

لنفرض المتتالية A والمتكونة من N من العناصر A[1],A[2],…,A[N] .
فان الفرز الاختياري يتم كما يلي :-
أولا نجد اصغر عنصر في المتتالية ونضعه في أول موقع ثم نجد ثاني اصغر عنصر بين عنصر المتتالية ونضعه في الموقع الثاني وهكذا حتى نهاية جميع العناصر .

Pass1. Find the location LOC of the smallest in the list of N elements
A[1],A[2],…,A[N] , and then interchange
A[LOC] and A[1]. Then A[1] is stored .

Pass2. Find the location LOC of the smallest in the sublist of N-1 elements
A[2] ,A[3] ,… ,A[N] , and then interchange
A[LOC] and A[2] . then
A[1] ,A[2] is stored , since A[1] ≤ A[2]

Pass N-1. Find the location LOC of the smaller
Of the elements A[N-1], A[N], and
Then interchange A[LOC] and A[N-1]
Then A[1],A[2],…,A[N] is sorted
Since A[N-1] ≤ A[N]



لنفرض ان المتتاليه A والتي تحتوي 8 عناصر
77 , 33 , 44 , 11 , 88 , 22 , 55 قم بفرزها اختياريا

	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K=1 , LOC=4	77	33	44	11	88	22	66	55
K=2 , LOC=6	11	33	44	77	88	22	66	55
K=3 , LOC=6	11	22	44	77	88	33	66	55
K=4 , LOC=6	11	22	33	77	88	44	66	55
K=5 , LOC=8	11	22	33	44	88	77	66	55
K=6 , LOC=7	11	22	33	44	77	77	66	88
K=8 , LOC=7	11	22	33	44	55	66	77	88
Sorted	11	22	33	44	55	66	77	88

MIN (A , K , N , LOC)

An array A is in memory. This procedure finds the location LOC of the smallest element among A[k] , A[k+1] , … , A[N]

1.set MIN=A[k] and LOC=k initializes pointers

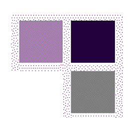
2.repeat for J=k+1 , k+2 , … , N

If MIN>A[J] then set MIN=A[J]

And LOC=A[J] and LOC=J

[end of loop]

3.return



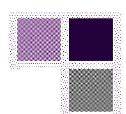
Selection (A,N)

This algorithm sorts the array A with N elements

1.repeat steps 2 and 3 for k=1 , 2 , … , N-1
2.call MIN(A,K,N,LOC)
3.[interchange A[k] and A[LOC]
  Set temp=A[k] , A[k]=A[LOC] and A[LOC]=temp
  [end of step 1 loop]
4.Exit

برنامج الاختيار :

```
Void sort ( inta[ ] , int n )
// sort the integers a [0…n-1] into no decreasing order
{
  Int I , j , k , t
  For ( i=0 ; icn ; i++ )
{
     J = I ;
// find smallest integer in a[ j…n-1 ]
    For(k=j+1 ; kcn ; k++)
       If (a[k] < a[j]
          J=k ;
// iuterchange
  T = a [ i ]
  A[ j ] = a[ I ]
A[ j ] = t
  } // end for i
}// end for sort
```



٣- الفرز الاختياري shell sort

سمي هذا النوع من أنواع الفرز بهذا الاسم نسبة إلى مصممه (D.L.shell)
وهو يعتبر فرزا ادخاليا معدلا.
وتعتمد هذه الطريقة على تقسيم عناصر البيانات إلى مجموعات حيث تصنف كل مجموعه بمفردها باستخدام الفرز الادخالي وبعد هذا تجمع هذه المجموعات معا وتفرز

فلو فرضت انه تم تقسيم المصفوفة في المرحلة الاولى الى k من القطاعات (حيث k تمثل عدد القوائم) حيث يفضل ان تكون قيمة k من القيم الاولية (prime number) وان عدد عناصر المصفوفة يساوي n
فان القوائم الجزئية التي تحصل عليها هي كالتالي :

القائمة الاولى A[1] , A[k+1] , A[2*k+1] , ...

القائمة الثانية A[2] , A[k+2] , A[2*k+2] , ...

.

.

.

القائمة k A[k] , A[2*k] , A[3*k] , ...

ملاحظه :- ان k تمثل عدد القوائم وبنفس الوقت مقدار بعد كل عنصر في قائمة ما عند العنصر الذي يليه في نفس القائمة

وبما انه يتم فرز كل قائمة جزئية على حد فانه بعد الجولة الأولى للخوارزمية تصبح القائمة مرتبة جزئيا.

وفي الجولة الثانية يتم اختيار قيمه جديدة لـk اقل من القيمة التي تم اختيارها في المرة الأولى وبذلك يزداد عدد العناصر في القوائم الجديدة ويقل عدد هذه القوائم عن التي تم الحصول عليها في الجولة الأولى.

وتستمر هذه العملية حتى تصبح قيمة k تساوي ١ ثم نطبق خوارزمية الفرز الادخالي مرة أخرى فنحصل على المصفوفة الاصليه مرتبة (مفروزة)

فائدة الفرز التجزئي :

أن ترتيب قائمة تبتعد عناصرها عن بعضها البعض k من العناصر يزيد من احتمالية أن يصل العنصر بسرعة إلى موقعه النهائي في المصفوفة . ولفهم ذلك تصور وجود عنصر كبير جدا في بداية المصفوفة A[1] هذا يعني انه في المرحلة الأولى سيقارن مع A[k+1] ثم مع A[2k+1] مما يعني انه سينتقل بسرعة إلى نهاية المصفوفة.

مثال:- افرض انك ترغب بترتيب القائمة التاليه باستخدام طريقة الفرز الجزئي.

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
80	32	38	19	36	45	21	16	12	6	40	7	72	90	50	15

المرحلة الأولى : تقسم القائمة إلى مجموعه من القوائم الجزئية عددها K فنختار K عددا أوليا على سبيل المثال K=7وعليه فان مكونات القائمة الأولى ستكون كما يلي:

A[1]=80

A[K+1]=A[1+7]=A[8]=16

A[1+2K]=A[1+14]=A[15]=50

A[1+3K]=A[1+21]=A[22]

أصبح هذا العنصر اكبر من حجم المصفوفة الرئيسية وهو ١٦ عنصر

وعليه القائمة الأولى تضم 80 , 16 , 50

عناصر المجموعة الثانية :

A[2]=32

A[2+K]=A[2+7]=A[9]=12

A[2+2K]=A[2+14]=A[16]=15

A[2+3K]=A[2+21]=A[23]

اكبر من حجم المصفوفه (نتوقف)

عناصر القائمة الثانية 32 , 12 , 15

عناصر المجموعة الثالثة:

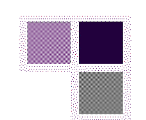
A[3]=38

A[3+K]=A[3+7]=A[10]=6

A[3+2K]=A[3+14]=A[17]

نتوقف اكبر من حجم المصفوفه

وعليه عناصر القائمة الثالثه 32 , 15 , 12



- وهكذا تستمر حتى نصل إلى القائمة السابعة K=7

- وعليه نحصل على :

القائمة الأولى	80 , 16 , 50
القائمة الثانية	32 , 12 , 15
القائمة الثالثة	38 , 6
القائمة الرابعة	19 , 40
القائمة الخامسة	36 , 7
القائمة السادسة	45 , 72
القائمة السابعة	21 , 90

يتم ترتيب كل قائمة من القوائم أربعه بواسطة الفرز الادخالي وبالتالي نحصل على القوائم مرتبة كما يلي :

الأولى	16 , 50 , 80
الثانية	12 , 15 , 23
الثالثة	6 , 38
الرابعة	19 , 40
الخامسة	7 , 36
السادسة	45 , 72
السابعة	21 , 90

- بناء عليه تصبح القائمة الكلية بعد الفرز الادخالي كما يلي :

16,12,6,19,7,45,21,50,15,38,40,36,72,90,80,32

المرحله الثانيه :

نختار قيمه أخرى لـK مختلفة واصغر من القيمة السابقة K=7 وعلى سبيل المثال اخترنا K=3 (حيث ثلاث عدد اولي)

عناصر القائمة الأولى ستبدو كما يلي

A[1]=16

A[1+K]=A[1+3]=A[4]=19

A[1+2K]=A[7]=21

A[1+3K]=A[10]=38

A[1+4K]=A[13]=72

A[1+5K]=A[16]=32

وعليه تتكرر حتى القائمة الثالثة K=3 نحصل على

القائمة الأولى 16 , 19 , 21 , 38 , 72 , 32

القائمة الثانية 12 , 7 , 50 , 40 , 90

القائمة الثالثة 6 , 45 , 15 , 36 , 80

بعد الترتيب الادخالي لكل قائمة على حدا نحصل على :

الأولى 16 , 19 , 21 , 32 , 38 , 72

الثانية 7 , 12 , 40 , 50 , 90

الثالثة 6 , 15 , 36 , 45 , 80

وعليه تبدو القائمة الكلية كما يلي

16,7,6,19,12,15,21,32,36,38,40,36,32,50,45,38,90,80,72

- نكرر عملية التجزئة حتى نصل الى قيمة K=1 وبتطبيق خوارزمية الفرز الادخالي وبعدها نحصل على القائمة الرئيسية مرتبة ترتيبا تصاعديا كما يلي:

6,7,12,15,16,21,32,36,38,40,45,50,72,80,90

ملاحظه :يجب ان تكون القيمة النهائية في SHELL SORT لـ K=1

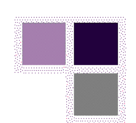
Procedure shell sort (A , K , Num) ← a عدد العناصر في المصفوفة

المصفوفه ← A ، عدد القوائم ← K

Set k= num/2 عدد اولي

Repeat while k>0

Call insertion algorithm



K=k/2 قلل قيمةk واعد الكرة
[end of loop]
Return

الفرز السريع quick sort

افرض لدينا المصفوفة A

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
8	22	30	16	20	18	12	14	25

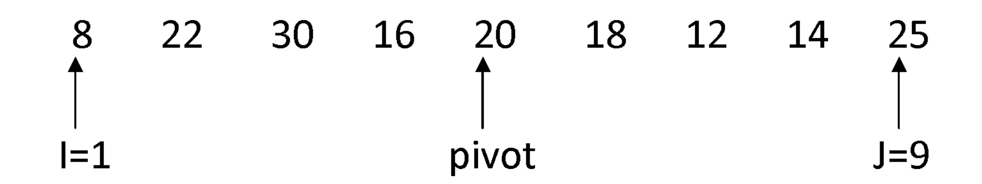
طريقة الفرز السريع :

١- نختار عنصر معين من عناصر المصفوفة بشرط ان قيمة هذا العنصر تتوسط مجموعة العناصر ويسمى هذا العنصر باسم القطب PIVOT

في المثال القطب نختاره A[5]=20

٢- نقوم بتعيين مؤشرين (J,I) للتأشير على عناصر المجموعة

حيث I يستعمل للتأشير على بداية القائمة أي أن القيمة المبدئية للمؤشر I=1 J يستعمل للتأشير على نهاية المصفوفة أي أن قيمة المؤشر في المثال J=9

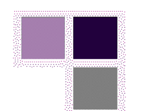


٣- نبدأ عملية مقارنة قيمة عنصر المصفوفة المشار إليه بالمؤشر I مع القطب والهدف هو إيجاد العناصر الأكبر من القطب ونقلها إلى اليمين من القطب

* وفي كل مقارنة إذا كانت قيمة العنصر اقل من قيمة القطب يتم تعديل قيمة المؤشر I بزيادة 1 له ليشير للعنصر التالي ويترك العنصر في مكانه .

* وتتوقف عملية المقارنة عندما تصل إلى عنصر قيمته اكبر من قيمة القطب وهذا يعني أن مثل هذه القيمة يجب أن تكون لليمين من القطب (أي ضمن العناصر التي تزيد قيمتها عن القطب)

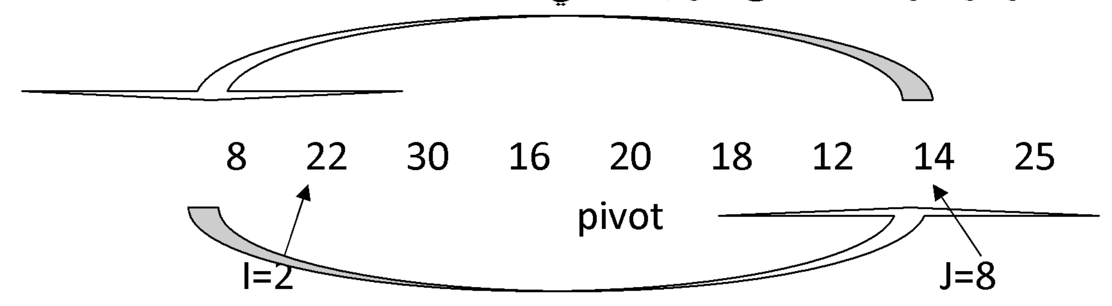
- في مثالنا تستمر المقارنة إلى أن نصل إلى العنصر 22 وهوا اكبر من القطب 20.



* ومن الجهة الأخرى للمصفوفة تبدأ عملية المقارنة بين العنصر المشار إليه بالمؤشر J مع القطب والهدف هو إيجاد العناصر الأقل من القطب وذلك لنقله إلى يساره وطالما أن المؤشر J يشير إلى عنصر اكبر قيمة من القطب فانه يتم تنقيص 1 من قيمة المؤشر وذلك للانتقال إلى العنصر السابق في هذا الجزء من العناصر

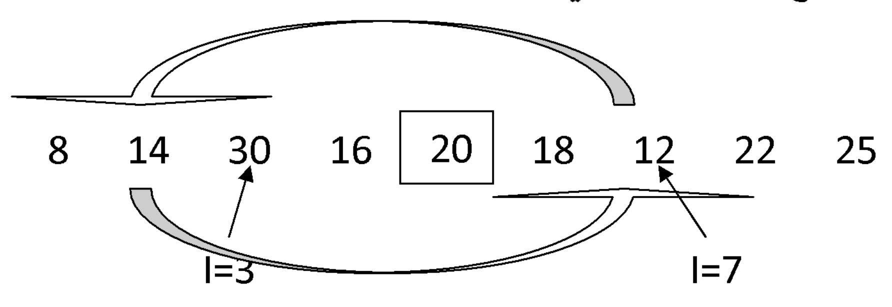
- في مثالنا :

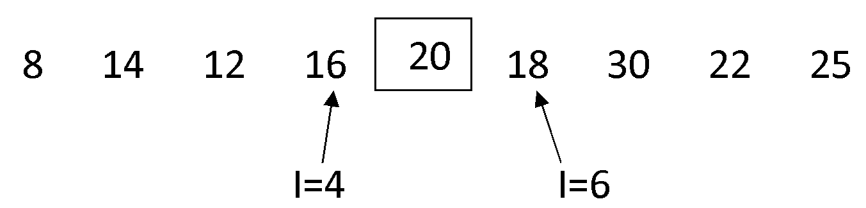
نلاحظ أن 25 اكبر من القطب 20 لذا يجب أن تبقى في مكانها وننقص 1 من J فنصل إلى 14 وبما أن 14 اقل من القطب 20 لذلك تتوقف عملية التنقيص من المؤشر J ونحصل على الترتيب التالي :



بما أن العناصر التي قيمتها اقل من القطب يجب ان تكون على يسار القطب والعناصر التي قيمتها اكبر من القطب على اليمين فانه يجب تبديل مواقع القيم 14 ,22

فتصبح المتتالية كما يلي





نقارن القيمة 16 مع القيم 20 وبما أن 16 اقل من 20 فنعمل على زيادة المؤشر I بمقدار 1 أي أن I=5 بحيث يصبح المؤشر I يشير إلى القيمة 20 . وتتم بعد ذلك مقارنة القيمة التي يشير إليها المؤشر وهي 20 مع القطب 20 وبما ان 20 ليست اقل من 20 فانه لا تطرأ زيادة على قيمة I وتتوقف عملية المقارنة بين القيم المؤشر إليها بالمؤشر I والقيمة 20 .

- ثم ننتقل إلى الجهة الأخرى من المصفوفة أي الجهة المؤشر J فبما أن المؤشرJ يشير إلى عنصر قيمته اكبر من 20 فيجب تنقيص1 من J . وبما أن المؤشر المشار إليها بالمؤشرJ (أي أن القيمة 18) ليست اكبر من 20 فانه لا يطرأ أي تنقيص على قيمة J . ولان تبدأ المبادلة بين العنصر 18,20 وتزداد قيمة I بمقدار1 أي تصبح I=6 وكذلك تنقص قيمة J بمقدار1 لتصبح J=5

8 14 12 16 18 20 30 22 25

J=5 I=6

- نلاحظ أن قيمة I أصبحت اكبر من قيمة المؤشر I وهذا يعني انتهاء الجولة الأولى . وعليها نحصل على قائمتين جزئيتين من يسار القطب 20 كل العناصر الأقل منه ، ومن يمين القطب نفس القطب وما هو اكبر منه أي نحصل على القوائم الجزئية التالية :

8 14 12 16 18

20 30 22 25

- نكرر العملية السابقة على القوائم الجزئية كلا على حدا :

8	14	12	16	18
I=1		pivot		I=5

8	14	12	16	18
	I=2	J=3	J=4	J=5
	اكبر نتوقف	تقارن مع نفسها	تبقى	تبقى

8	12	14	16	18
	J=2	I=3		

J < I

بما أن J<I فأننا نحصل على قائمتين جزئيتين جديدتين و هما :

8 12 غير مرتبه

14 16 18 مرتبه

نأخذ القائمة الغير مرتبة القائمة اليمنى القائمة اليسرى

12 8

بما أنها تتكون من عنصرين فلا داعي للفرز وإنما نستبدلهما بالمواقع

بعد نأخذ القائمة الجزئية الثانية و هي

20	30	22	25
I=1		pivot	J=4

20	30	22	25
	I=2		J=4
	اكبر نتوقف		تبقى

20 30 22 25

I=2 J=3

20 22 30 25

J=2

نحصل على قائمتين

مرتبة 22 20

غير مرتبة 25 30 نبدل بينهما دون فرز لأنهما من عنصرين فنحصل 30 25

وبالتالي نحصل على المصفوفة مرتبة كاملا

8 12 14 16 18 20 22 25 30

مثال / افرز المصفوفة التالية :

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
7	23	29	15	19	18	13	14	24

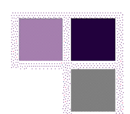
Pivot

↑

([1]+[9])/2=[5]

خوارزمية الفرز السريع :

```
PROCEDURE (A: array type, N: number of element in the array)
  {recursive quick procedure to sort an array A of integer
numbers in an ascending order }
     First: integer
     Last: integer
PROCEDURE Qsort (first, last)
     I,J   { array indices }
     Pivot   { middle array value }
     Temp: integer {temporary variable}
BEGIN
  I=first
  J=last
{pivot  تحديد القطب }
Pivot = A [(first+last)/2]    {find middle array value}
Repeat
  While A[I]<pivot   do
     I=I+1
  While A[J]<pivot   do
    J=J-1
If  I <= J THEN
BEGIN       {switch A[I] with A[J]
 Temp =A[I]
 A[I] = A[J]
A[J] = temp
I = I+1
J = J-1
END
Until I>J
```

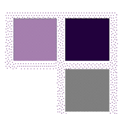


{ اذا احتوت القائمة اليسرى على أكثر من عنصر استدعي Qsort ذاتيا لترتيبها }

```
If first < J then
  Qsort ( first , J )  { make recursive call to Qsort with same first
                         value and last value set to J }
```

{إذا احتوت القائمة اليسرى على أكثر من عنصر استدعي Qsort ذاتيا لترتيبها }

```
If  I < last  THEN
  Qsort ( I , last )
END {Qsort}
BEGIN {Quick sort}
   Qsort {1 ,N}
END {quick sort}
```



الفرز المقيد heap sort

سمي هذا الفرز بهذا الاسم لاستخدامه البيانات الممثلة على شكل شجيرة ثنائية ذات خواص معينه (مقيده).
لذلك قبل إجراء هذا الفرز لابد من تحويل العناصر المراد فرزها والتي تكون ممثله على شكل مصفوفة إلى شجيرة ثنائية مقيده يطلق عليها اسم الشجيرة الثنائية المقيده (heap) والتي تتميز بالخواص التالية :

١- يجب أن تكون الشجيرة ثنائية كاملة .
٢- يجب أن تكون القيمة المخزنة ي كل موقع (node) في مواقع الشجيرة اصغر من القيمة المخزنة في الموقع الذي يشكل الأهل (parent) لهذا الموقع او مساوية لها.

على سبيل المثال :

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
16	19	14	10	15	7	20	8	12	18

ملاحظه :

كما نعلم لو فرضنا انه عنصرا ما يقع في الموقع2I وكذلك الابن الايمن (ان وجد) سوف يقع في الموقع (2I+1) وهذا نابع من تعريف الشجيرة الثنائية الكاملة عند تمثيلها في مصفوفة كما هو في المثال الحالي .

وتتم عملية إنشاء الشجيرة المقيدة داخل المصفوفة وذلك حسب الخطوات التالية:-

١. خذ العنصر الأول والذي يحتوي القيمة 16 واعتبره شجيرة ثنائية مقيده تحتوي على عنصر واحد قيمته (16) .
٢. كون شجيرة ثنائية مقيده من العنصرين الأول والثاني في المصفوفة أي القيمتين (16 , 19).
٣. كون شجيرة ثنائية مقيده من العناصر الثلاثة الأولى في المصفوفة وهي التي تحتوي القيم (16 , 19 , 14).
٤. كرر هذه العملية وذلك بإضافة عنصر جديد إلى الشجيرة المقيدة في كل خطوه ، حتى تحصل على شجيرة ثنائيه مقيده تحتوي على جميع العناصر المطلوب الفرز عليها

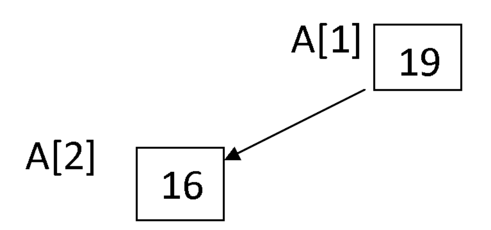
١- في البداية يتم إنشاء شجيرة ثنائية مقيده تحتوي على عنصر واحد وهو A[1]=16 حسب المثال ويعتبر هو الجذر لهذه الشجيرة.

٢- في الخطوة الثانية نأخذ الثاني التالي A[2]=19 ونكون منه مع A[1]=16 شجيرة ثنائية مقيده

A[1]= 16 A[2]= 19

ولكن يجب أن نحافظ على خواص الشجيرة المقيدة (أي أن قيم الأبناء اصغر أو مساوية لقيمة الأهل) وبما أن A[2] هو الابن الوحيد لهذه الشجيرة فيجب تبديل هذين العنصرين مع بعضهما البعض لان 19>16

وبهذا نحصل على الشجيرة الثنائية المقيدة

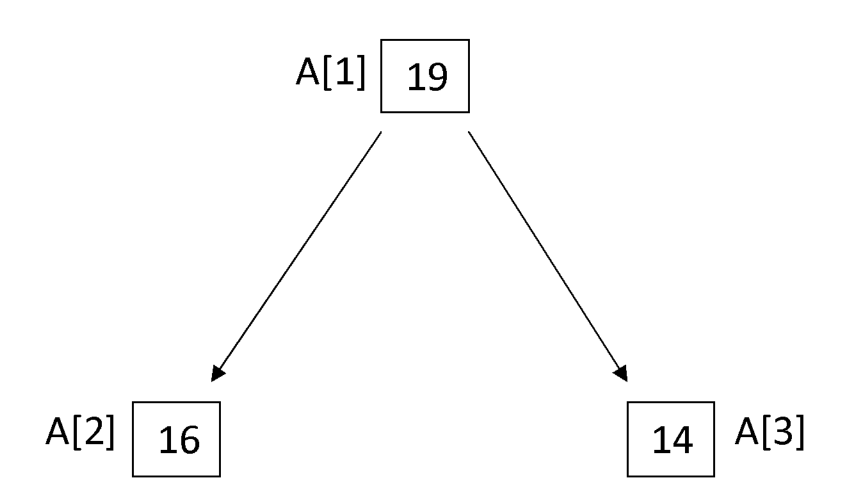


٣- في الخطوة الثالثه ناخذ A[3]=14

بما ان 19>14 فهو ايضا ابن لـA[1]

- ملاحظه / A[2] حسب الموقع ناتجة عن A[2i] = A[2*1] = A[2]

اما A[3] = A[2i+1] = A[2*1+1] فهو الابن الأيمن وبهذا تصبح الشجيرة كما يلي



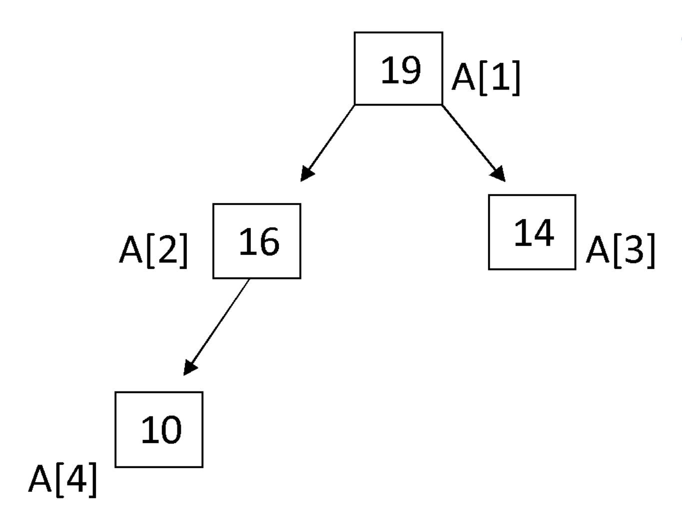
٤- وفي الخطوة الرابعة نضيف العنصر التالي A[4]=10 ونجري التعديل اذا لزم الأمر.

نجد أن A[4]=10

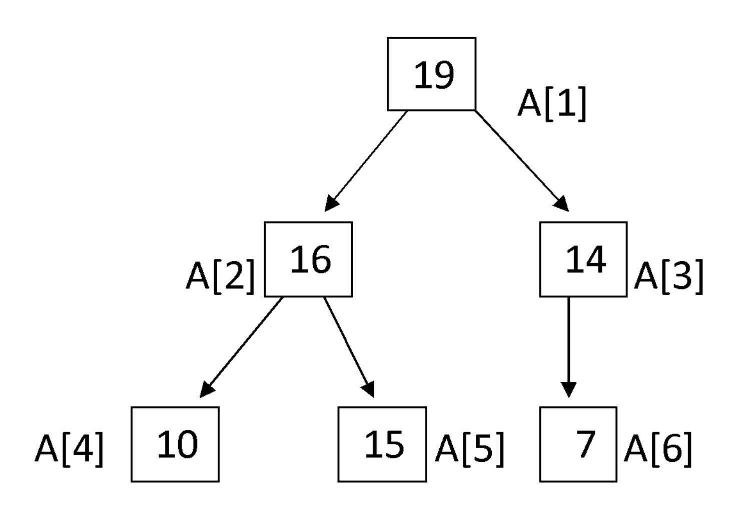
١) هي ابن أيسر لـA[2] وهذا لأنه A[4] = A[2*i] = A[2*2]

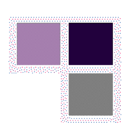
٢) أن 16 > 10 ولذلك لا يتم أي تبديل.

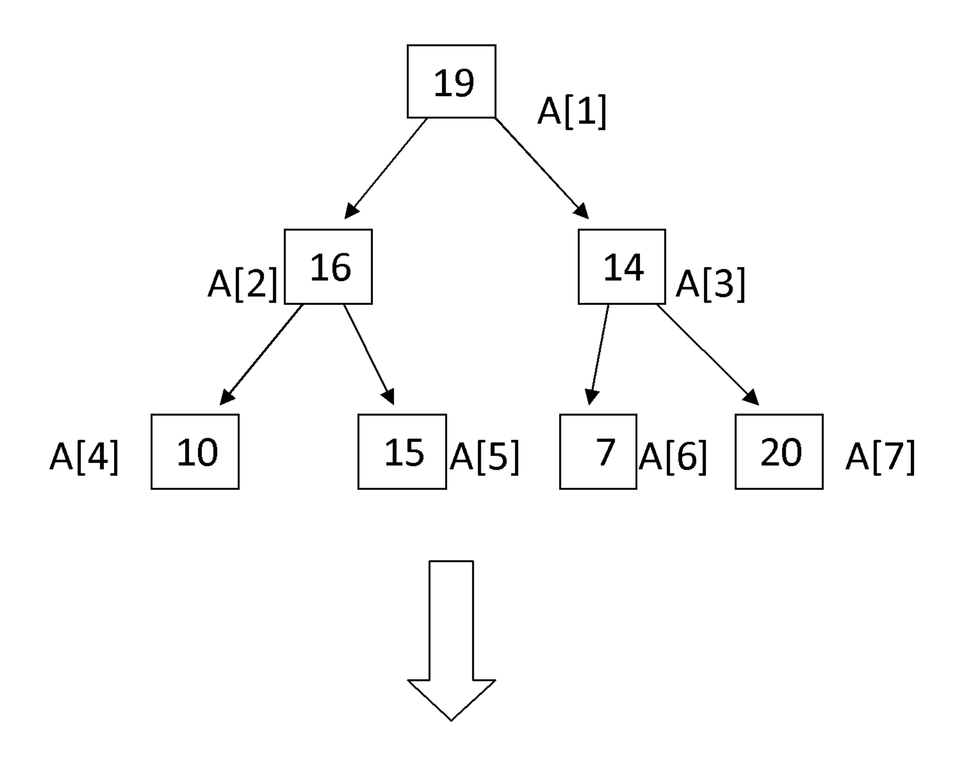
في الناتج نحصل على :

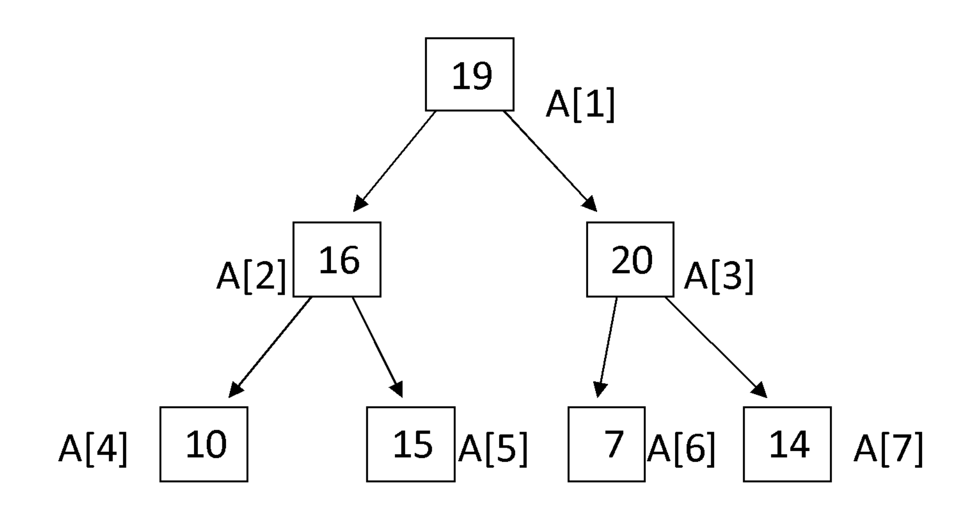


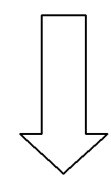
٥) نكرر العمليات أعلاه وهذا باضافه عنصر جديد في كل مرة مع إجراء المبادلة (التعديل) عند لزوم الأمر للمحافظة على خاصية الشجيرة الثنائية المقيدة.

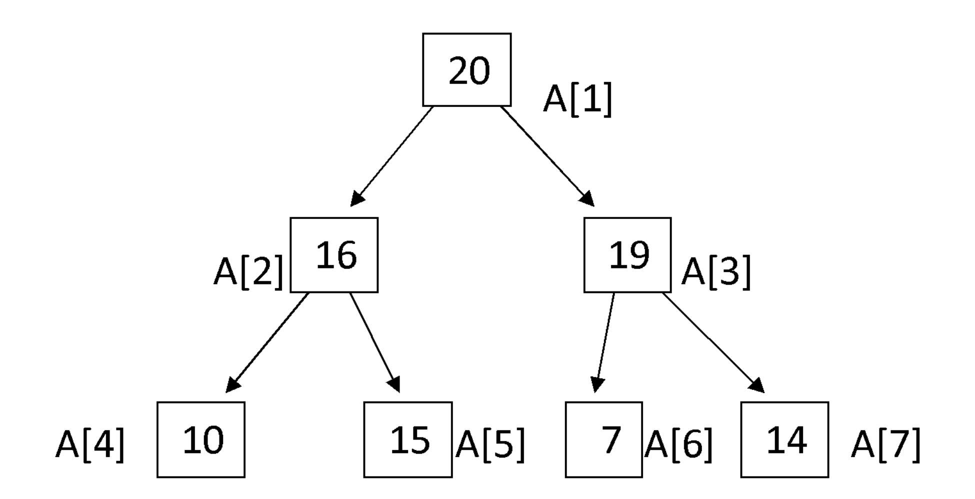


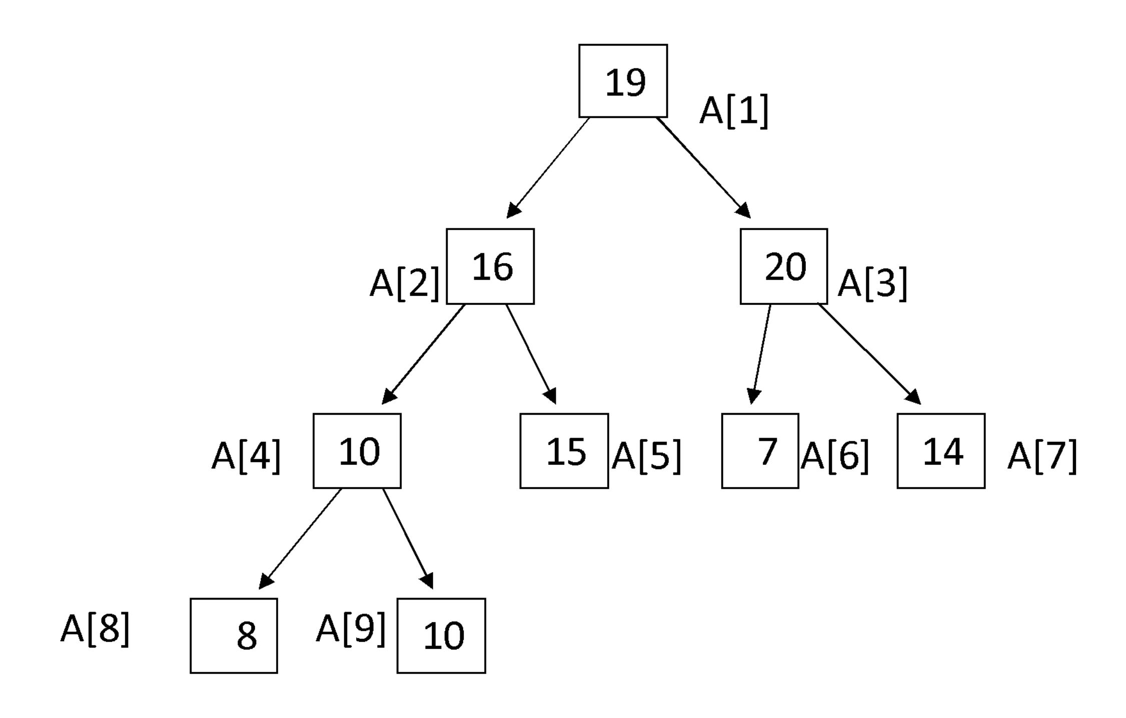



19
A[1]
16
A[2]
14
A[3]
A[4]
10
15
A[5]
7
A[6]
20
A[7]

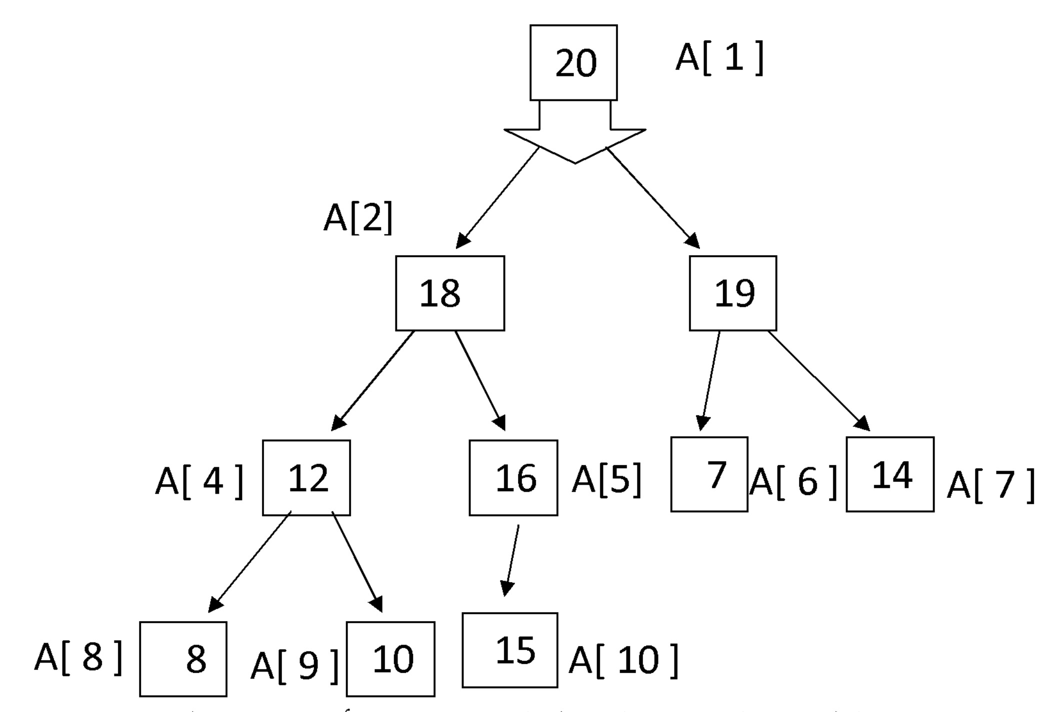

19
A[1]
16
A[2]
20
A[3]
A[4]
10
15
A[5]
7
A[6]
14
A[7]




20
A[1]
A[2]
16
19
A[3]
A[4]
10
15
A[5]
7
A[6]
14
A[7]


19
A[1]
A[2]
16
20
A[3]
A[4]
10
15
A[5]
7
A[6]
14
A[7]
A[8]
8
A[9]
10

- نضيف A[10] وهي A[10] = A[2i] = A[2*5] أي انها ابن ايسر لـA[5] فنحصل على الشجرة التالية :

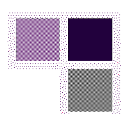


بعد الانتهاء من عملية بناء الشجيرة الثنائية المقيدة نلاحظ أن اكبر قيمة موجودة في العنصر A[1]=2 وهذا بشكل نهائي المرحلة الأولى :

خوارزمية الفرز المقيد للمرحلة الأولى

```
PROCEDURE Build heap
I,J , K    {indices for nodes
For  K=2  to N { N: number of elements in the array}
I=K {initialization I to index anode}
J=I/2 {initialize J as an index to parent node }
While(I<>1) AND (A[J]<A[I])
Item = A[J] {switch contents of node I }
A[J]=A[I] {with that of its }
A [I]= item {parent }
I=J     { set I to point to its parent }

IF I>1  then
J = I /2   { set J to point to ints parent }
END
END
END {procedure Build heap}
```

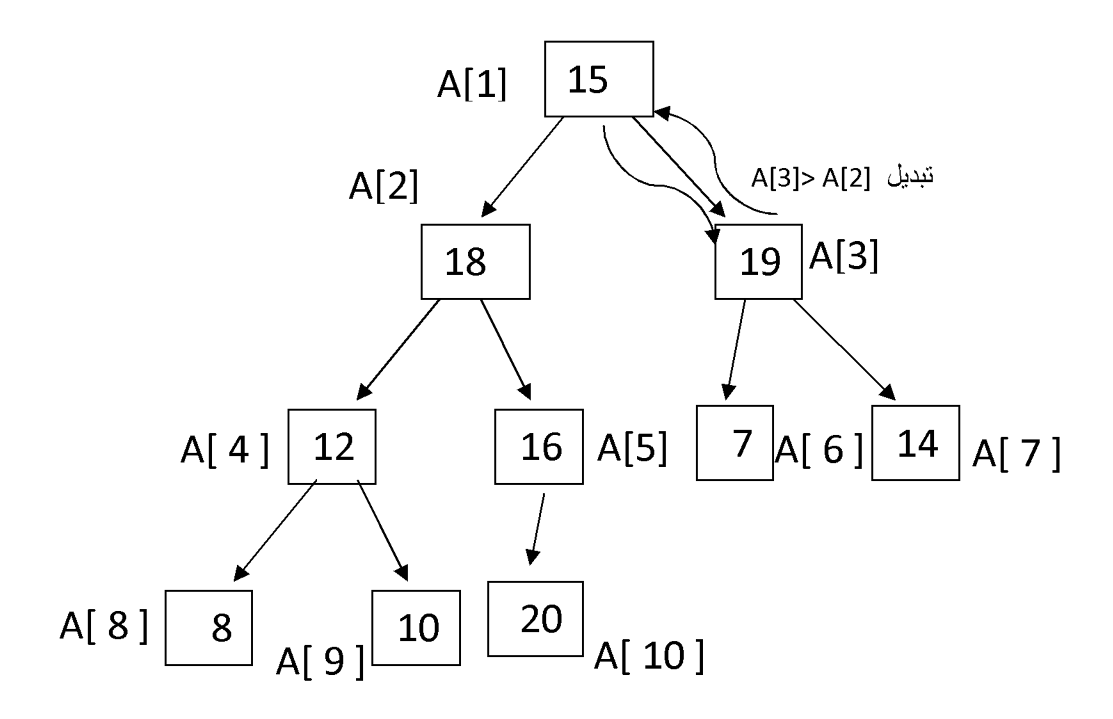


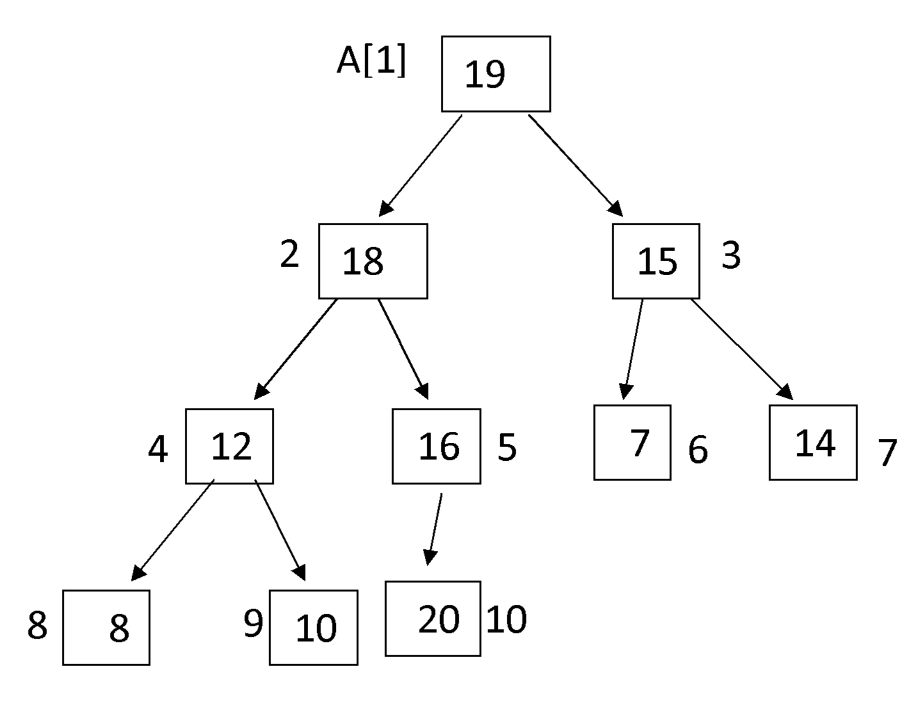
- في المرحلة التالية نقوم بتبديل العنصر الأكبر في الموقع الأول من المصفوفة والذي يمثل الجذر مع العنصر الموجود في الموقع الأخير من المصفوفة :
- وفي مثالنا : A[1]=20 نبدل مع A[10] =15

وبالتالي تصبح المصفوفة كما يلي :

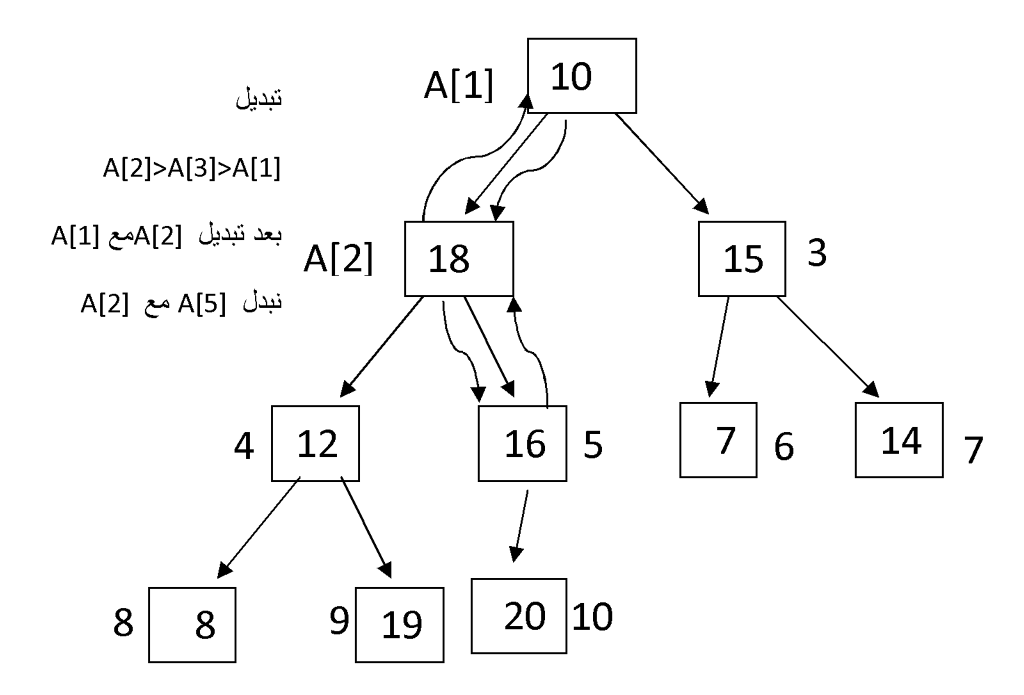
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
15	18	19	12	16	7	14	8	10	20

- ثم نستثني أخر عنصر وهو A[10]=20 لأنه اكبر عنصر أصبح ونقوم ببناء شجيرة ثنائية مقيدة مكررا ولكن بالمقارنة ابتداء من A[1]إلى A[g] فقط



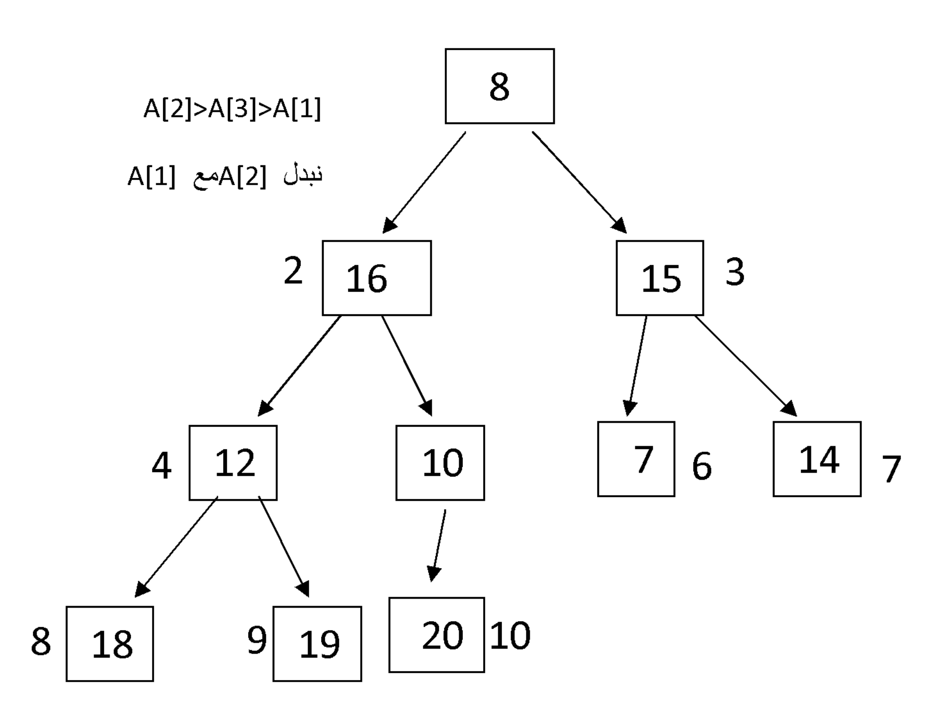


نبدل قيمة الجذر A[1]=19 مع A[g] فنحصل



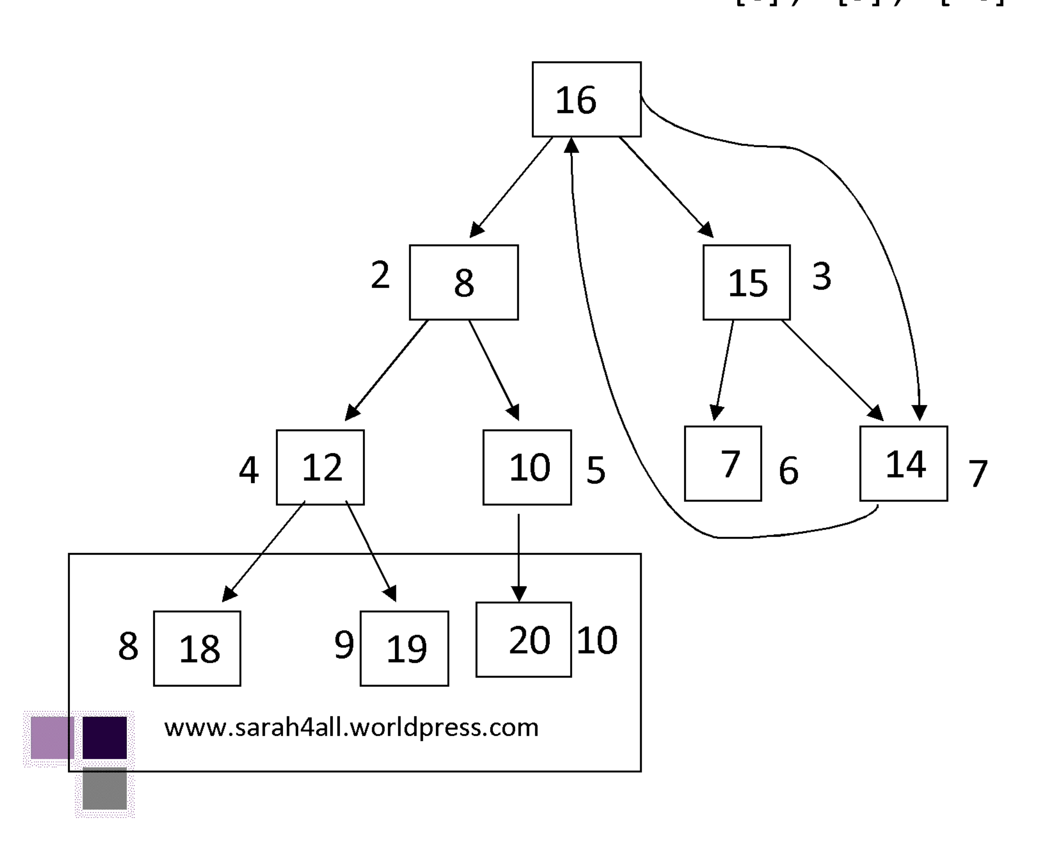
نبني شجيرة مقيدة باستثناء [9]A[10] , A

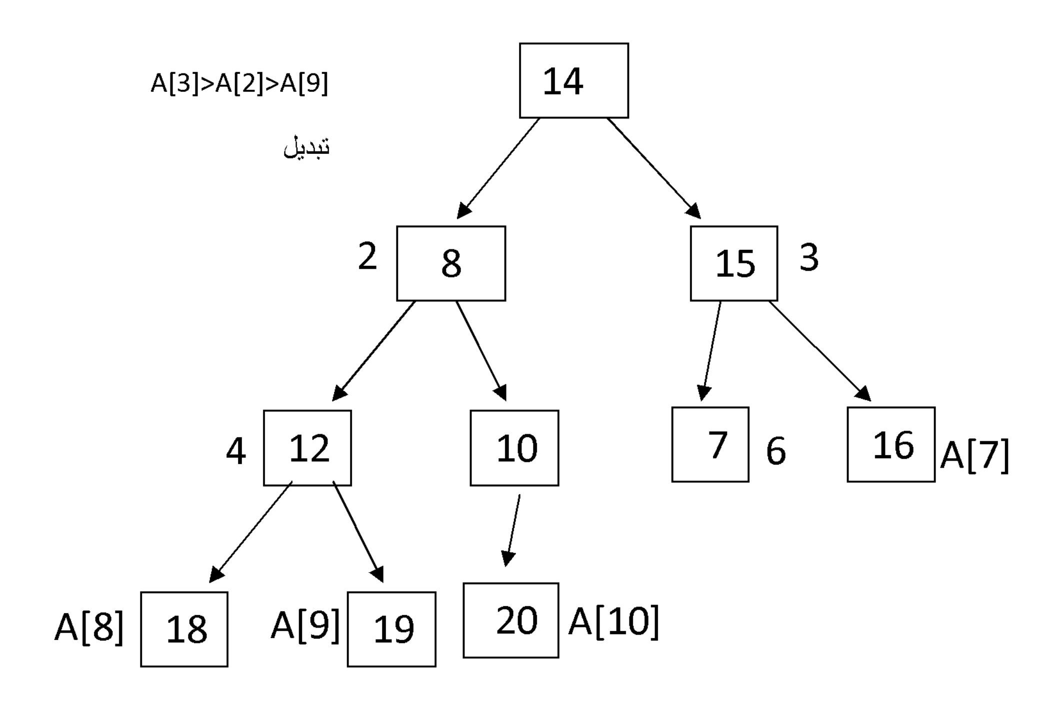
أي من [1]Aإلى [8]A

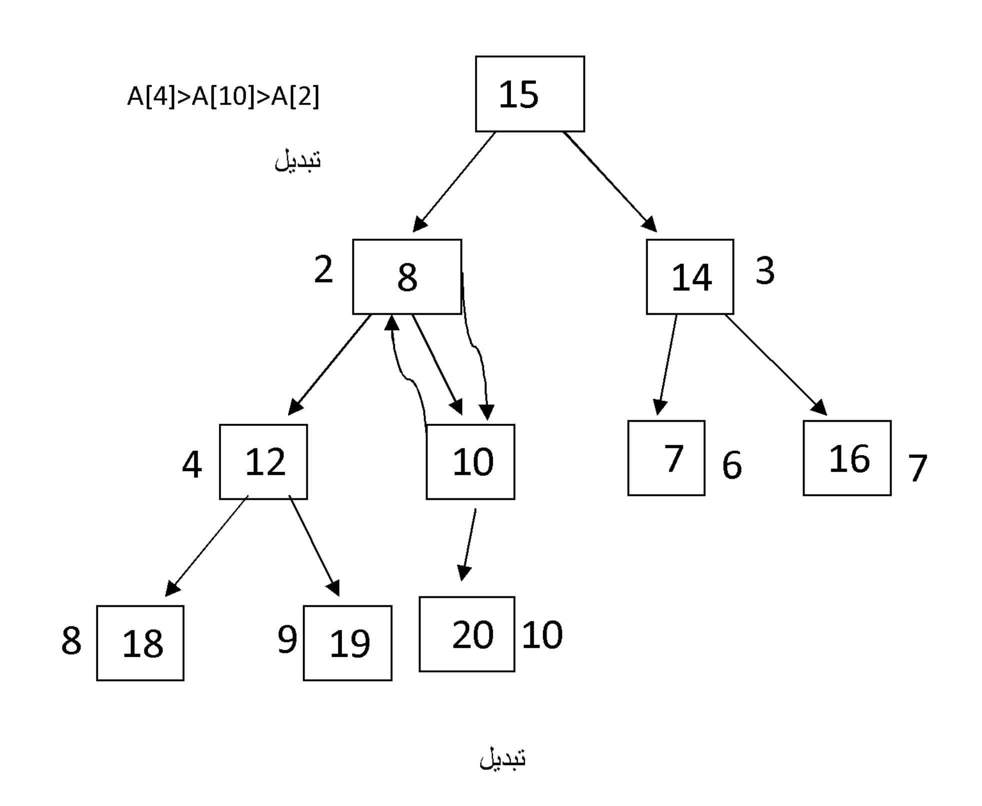


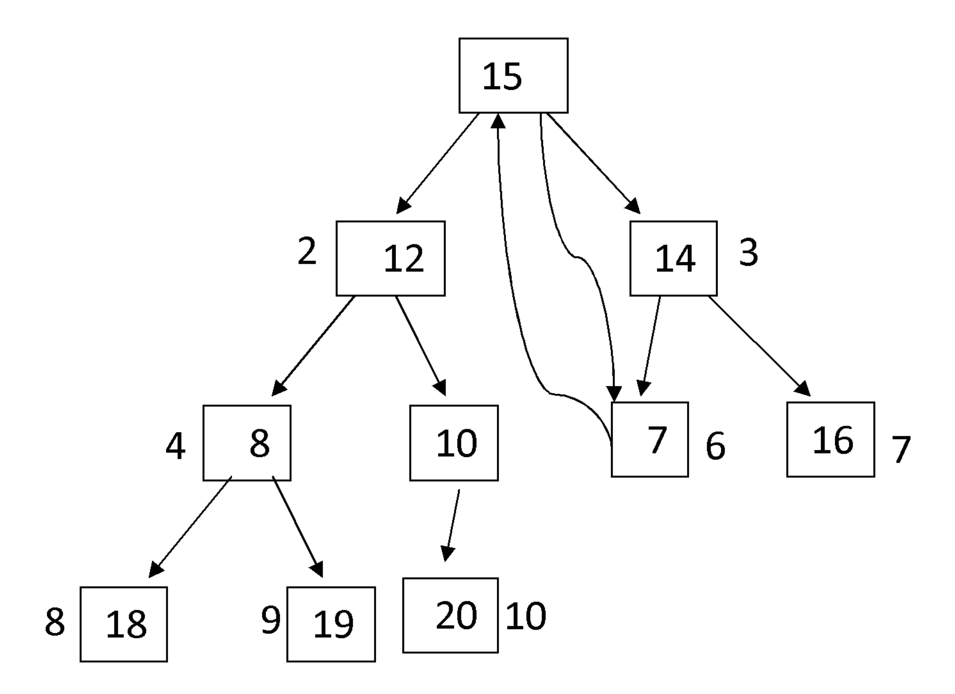
نبني شجيرة مقيدة من [1]Aإلى [7]A

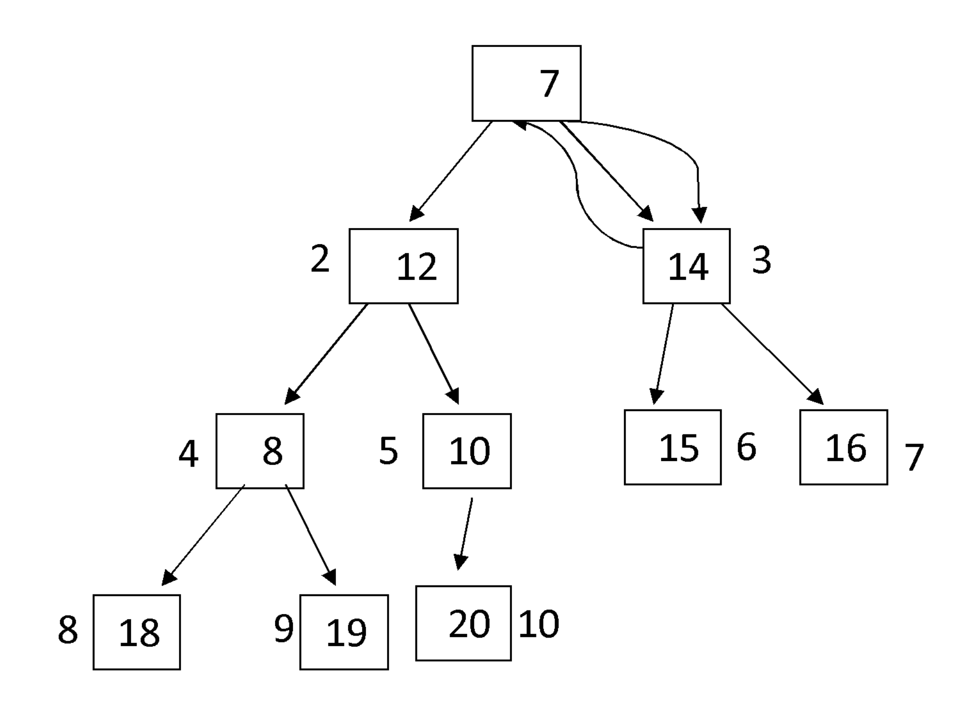
باستثناء [10]A[8] , A[9] , A

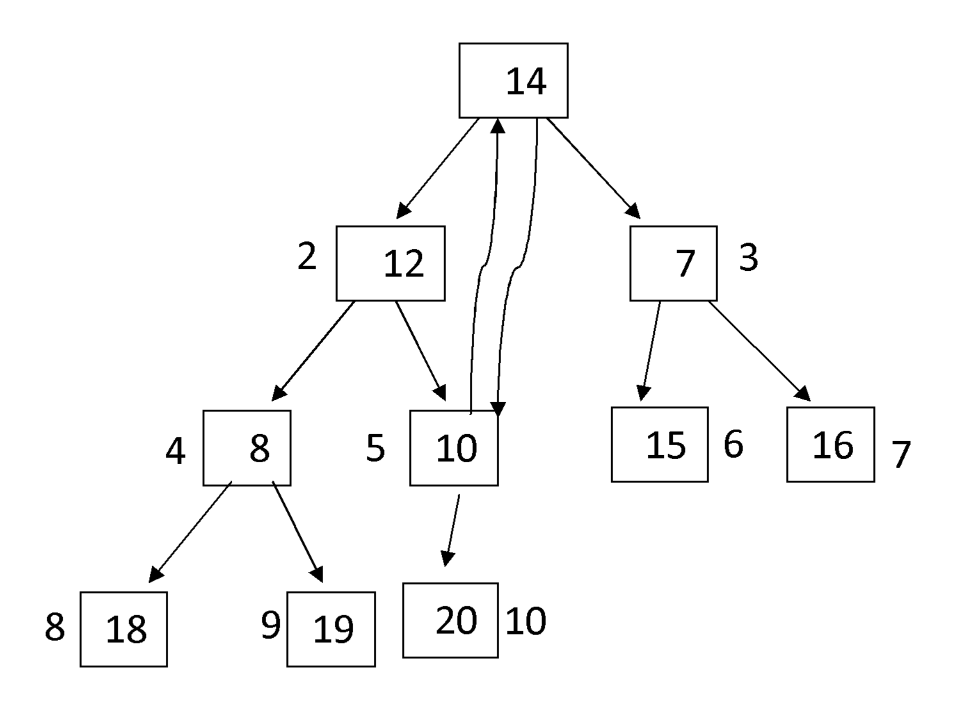


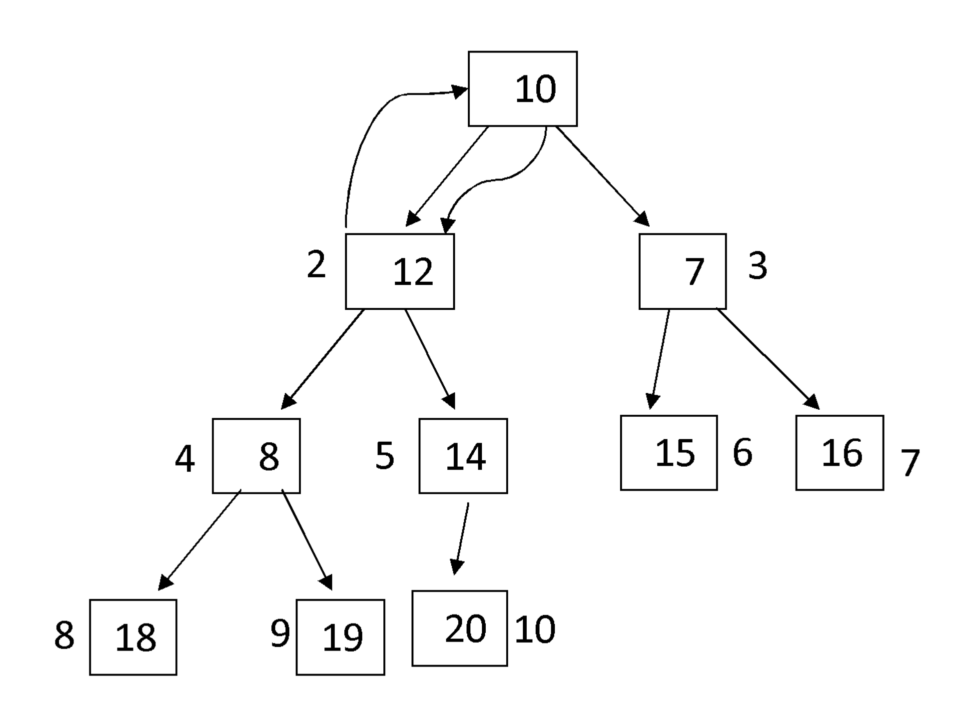

A[3]>A[2]>A[9]
تبديل
14
2
8
15
3
4
12
10
7
6
16
A[7]
A[8]
18
A[9]
19
20
A[10]

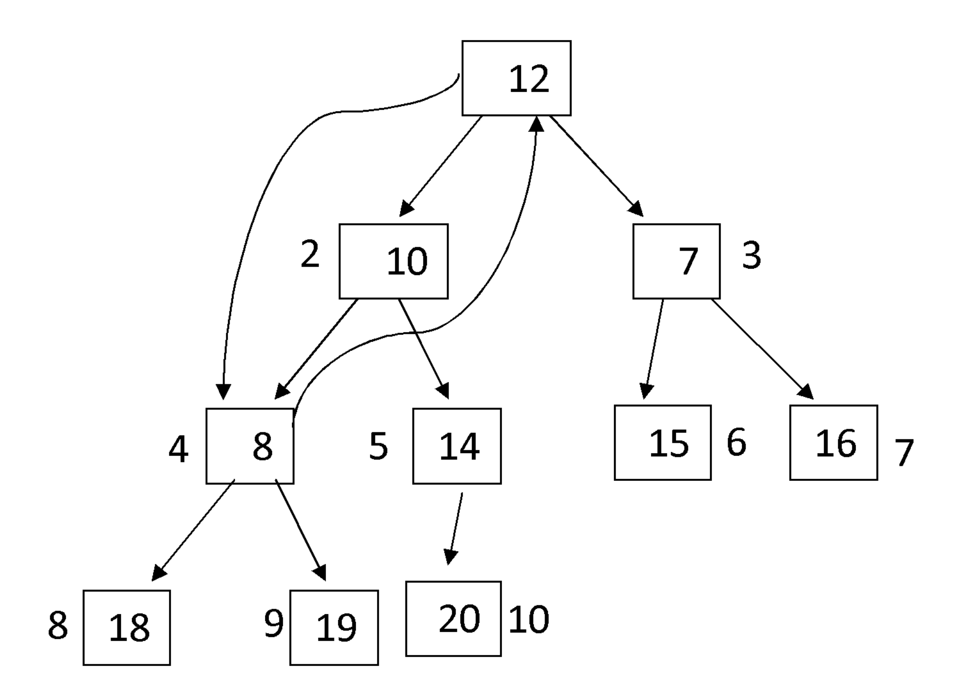

A[4]>A[10]>A[2]
تبديل
15
2
8
14
3
4
12
10
7
6
16
7
8
18
9
19
20
10
تبديل

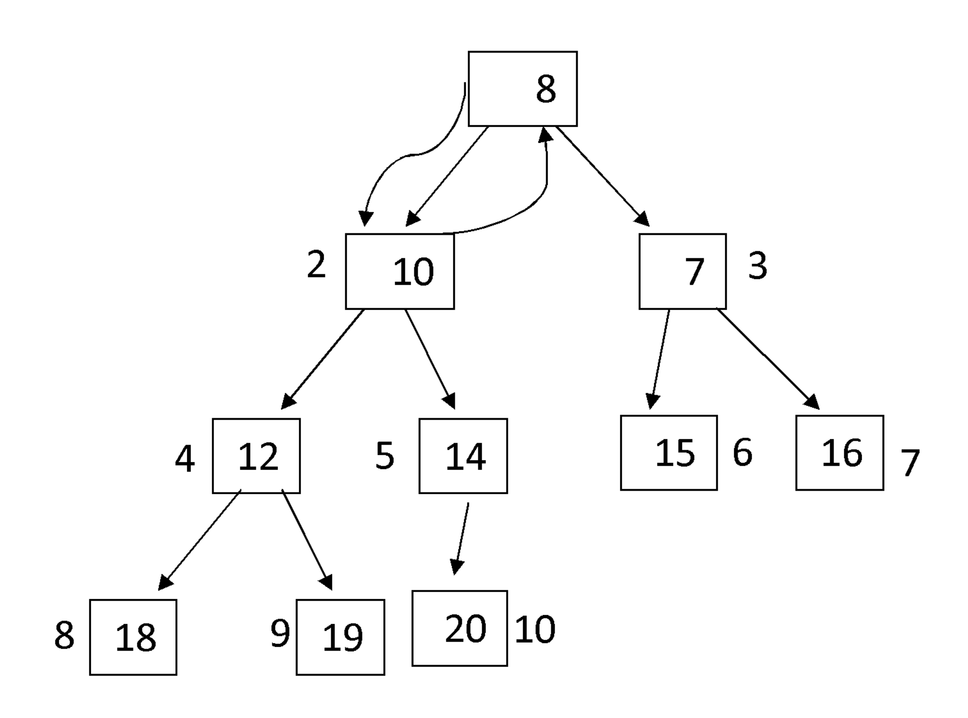

15
2
12
14
3
4
8
10
7
6
16
7
8
18
9
19
20
10

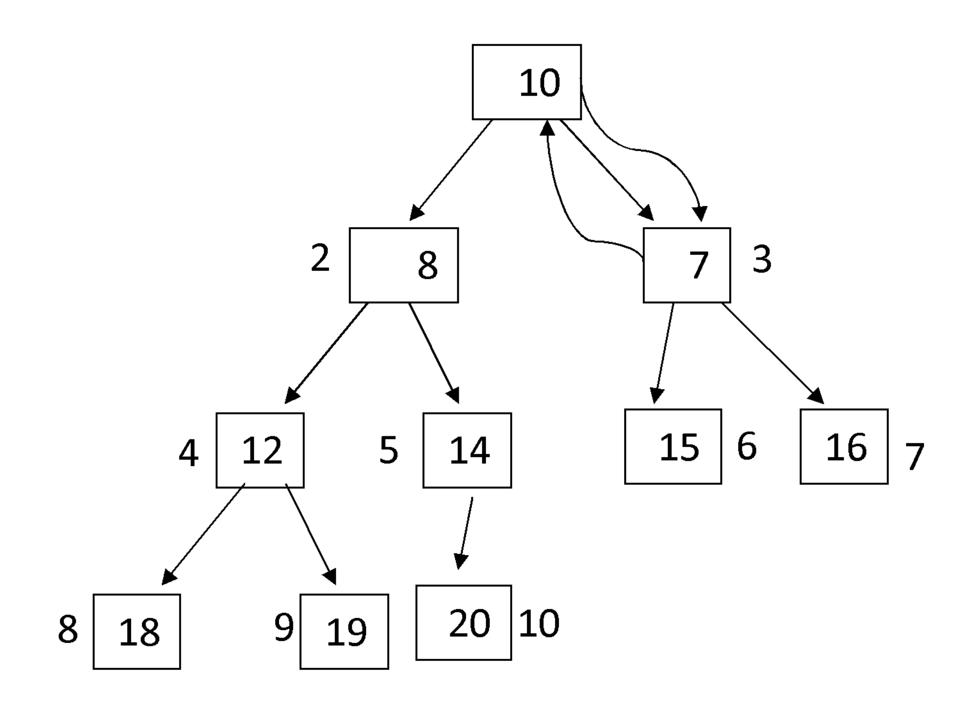

7
2
12
14
3
4
8
5
10
15
6
16
7
8
18
9
19
20
10

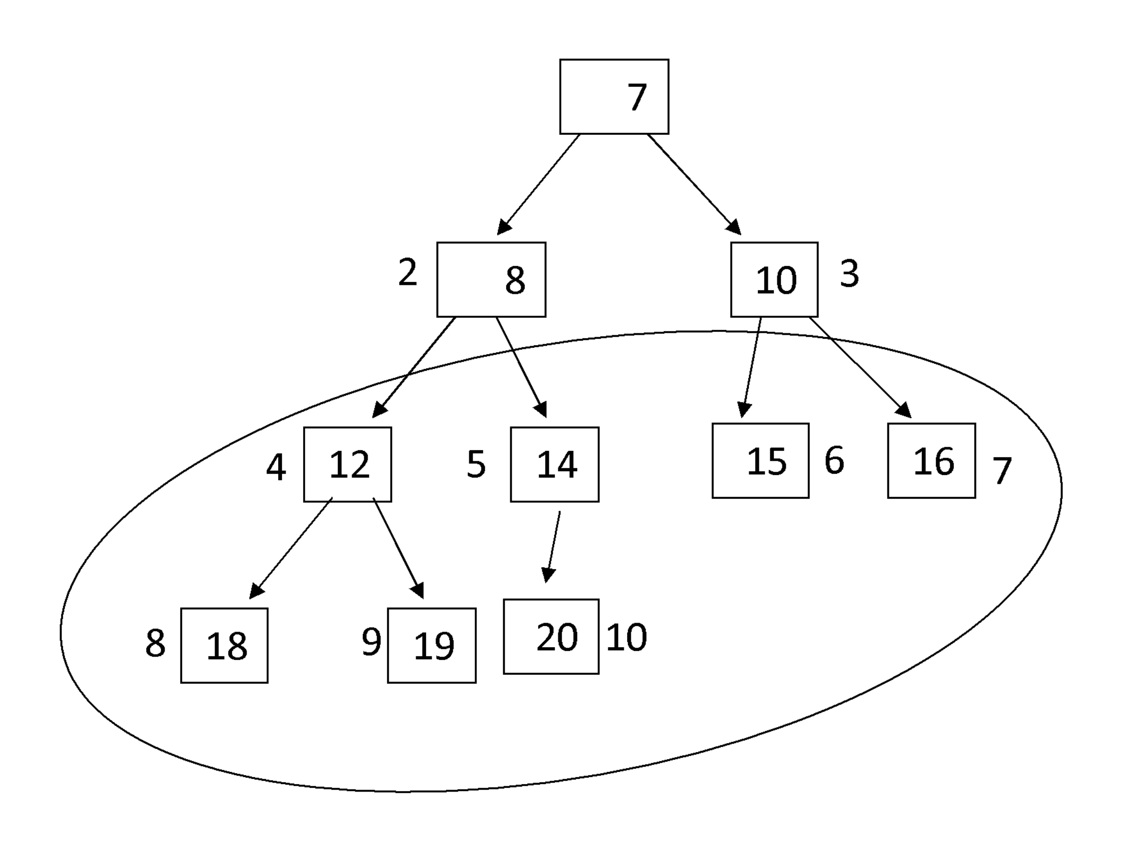

14
2
12
7
3
4
8
5
10
15
6
16
7
8
18
9
19
20
10


10
2
12
7
3
4
8
5
14
15
6
16
7
8
18
9
19
20
10


12
2
10
7
3
4
8
5
14
15
6
16
7
8
18
9
19
20
10


8
2
10
7
3
4
12
5
14
15
6
16
7
8
18
9
19
20
10


10
2
8
7
3
4
12
5
14
15
6
16
7
8
18
9
19
20
10


7
2
8
10
3
4
12
5
14
15
6
16
7
8
18
9
19
20
10

وبالتالي حصلنا على الترتيب التصاعدي للمصفوفة ...

- يمكن كتابة إجراء التعديل على الهيكل الشجري لتحويله الى هيكل شجري كامل مقيد (بحيث تبقى العناصر ممثلة في مصفوفة)كما يلي :

This procedure adjusts the nodes with indices from1 through M-1 so that the nodes again from a heap.

M: index of last node that was put in to its final position.

PROCEDURE Adjust Heap (M)

Element to move: integer {index of element to move

I {index

S: integer {child index

Done = Boolean {true when A [1] was not switched

Start

Element to move =A [1]

Done=false

I=1 {initialize I to root

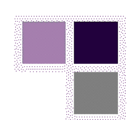
S=2 {initialize S to left child of root

While(s<M) and (NOT Done) Do

If(s+1) <M Then

S=S+1

If element to move>=A[s] Then



```
Done= true
Else
   {switch contents of node I with that of its child
A [I] =A[S]
A[S] = element to move
I=S      {set I to index to child
S= 2*I {set S to index to left child of new node I
END
END
END
```

والآن نكتب خوارزمية البرنامج الكامل للفرز المقيد والذي يستدعي الأجرائين السابقين..

N=100 {number of element in the array to be sorted

ArrayType=ARRAY [1…N] of integer

{Heap sort-Procedure to sort an array of integer Numbers in an ascending order using heap sort}

{Parameters

{A: Entire integer array to be sorted

{N: Number of elements in data Array

Procedure Heapsort (A, N)

Item: integer {an array element

J: integer {index

Insert Build Heap and Adjust Heap procedures here

Start

Build Heap {create the initial N-element heap

For J=N down To 2 Do {exchange element

Item=A [J]   J with root

A [J=A [1]

A [1] =Item

Adjust Heap (J)   {Adjust elements 1 through

J-1 so they reform

A heap of J-1 elements

End

End (heap sort)

☺ طرق الفرز الخارجية

External Sorting Methods

من أهم طرق الفرز الخارجي طريقة الفرز بالدمج

Merge sort

يستخدم هذا النوع من الفرز عند الحاجة لفرز كميه كبيرة من البيانات في ملف على أحد وسائط التخزين الخارجية.

ويتكون الملف عادة من عدد كبير من السجلات.ويمكن تلخيص خطوات الفرز بالدمج كما يلي

١ – يتم تقسيم الملف الرئيسي

F→F1, F2

٢ – بعد ذلك تتم مقارنة الملفين بالأخر بحيث نقارن عنصر من F1مع عنصر أخر في F2 . ومن ثم يتم دمجها حسب الترتيب المطلوب.

مثال :-

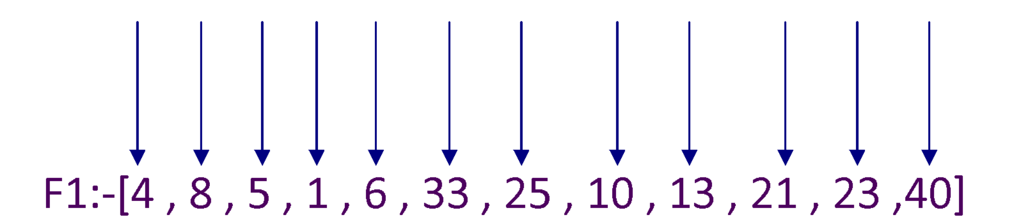
أفرض أنه معطى ملفا مخزنا على أحد أوساط التخزين الخارجي ويحتوي على البيانات التالية.

[4,8,5,1,6,33,25,10,13,21,23,40,16,60,30,47,32,11,37,14, 15,20,7,90,70]

وترغب بترتيبها تصاعديا.

الخطوة الأولى :

نجد ان العناصر في الملف الأصلي ٢٥ عنصر وعليه نقسم الملف الأصلي الى ملفين F1 ويحتوي على العناصر الأثني عشر الأولى و F2 ويحتوي على العناصر المتبقية وعددها ١٣ عنصر .

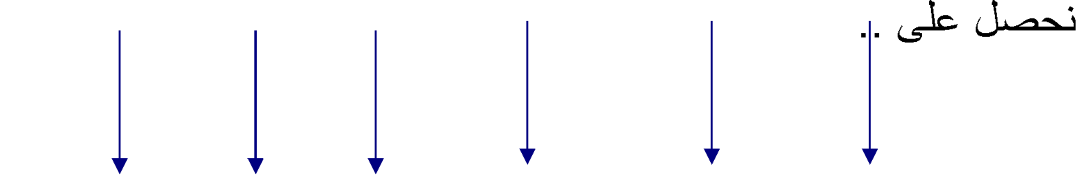


F1:-[4 , 8 , 5 , 1 , 6 , 33 , 25 , 10 , 13 , 21 , 23 ,40]

F2:-[16,60,30,47,32,11 , 37 , 14 , 15 , 20 , 7 , 90 , 70]

الخطوة الثانية :

يتم أخذ عنصر آخر من F1 مع عنصر من F2 لتكوين زوج جديد من العناصر ، بحيث يكون كل زوج من العناصر مرتبا . وبالاسلوب نفسه يتم تكوين جميع الزواج المرتبه من العناصر ويتم كتابة هذه الأزواج من العناصر بالتناوب الى ملفين جديدين يمكن تسميتها A1 و A2 وعندها نحصل على :.

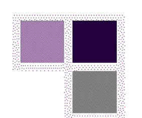


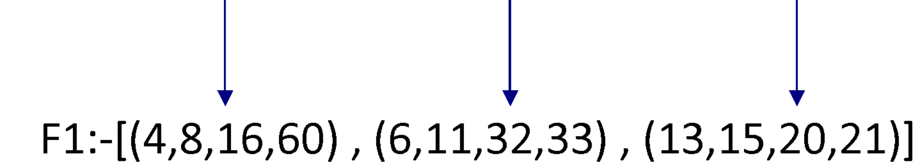
A1:-[(4,16),(5,30),(6,32) , (25,37) , (13,15) , (7,23)]

A2:-[(8,60),(1,47),(11,33),(10,14) , (20,21) , (40,90) , 70]

الخطوة الثالثة:

يتم تكوين قطاعات طول كل منها ٤ عناصر وذلك باخذ زوج من A2 وبالتناوب وتكوين قطاع طوله ٤ عناصر مرتبه ، وبعد ذلك نكتب هذه القطاعات الى الملفات F1 و F2 كما يلي :





F1:-[(4,8,16,60) , (6,11,32,33) , (13,15,20,21)]

F2:-[(1,5,30,47) , (10,14,25,37),(7,23,40,90) , 70]

الخطوة الرابعة :

يتم تجميع كل ٨ عناصر بحيث تؤخذ أربعة عناصر من الملف F1 وتدمج مع اربعه عناصر من الملف F2 وتكتب العناصر الثمانية مرتبة الى الملفين A1 و A2 بالتناوب .. فنحصل على :-

A1:-[(1,4,5,8,16,30,47,60) , (7,13,15,20,21,40,90)]

A2:-[(6,10,11,14,25,32,33,37) , 70]

الخطوة الخامسة :

يتم دمج القطاع الأول من A1 (والذي يحتوي على ٨ عناصر) مع القطاع الأول من A2 (والذي يحتوي أيضا على ٨ عناصر)

ويتم كتابة العناصر الناتجة إلى الملف F1

وكذلك يتم دمج القطاع الثاني (والذي يتكون من ٨ عناصر في A1 مع العنصر الوحيد في القطاع الثاني من A2 للحصول على قطاع طوله ٩ عناصر في الملف F2 بحيث تكون عناصره مرتبة

F1[1,4,5,6,8,10,11,14,16,25,30,32,33,37,47,60]

F2[7,13,15,20,21,23,40,70,90]

الجولة الخامسة

يتم دمج القطاع في F1 مع القطاع F2 بشكل مرتب والناتج يكون مرتب تصاعديا ويكتب في الملف A1 وعندها يصبح الملف A2 فارغا ..

وعندها تتوقف عملية الدمج ونحصل على ..

A1:-[1,4,5,6,7,8,10,11,13,14,15,16,20,21,23,25,30,32,33,37, 40, 47, 60, 70, 90

MERGING (A, R, B, S, C)

Let A and B be sorted arrays with R and S elements, respectively .This algorithm merges' A and B in to an array with N=R+S elements.

1. [initial2e] set NA=1, NB=1 and PTR=1

2. [compare] Repeat with Na ≤ R and NB ≤ s

If A [NA] <B [NB], then

A) [Assign element from A to C] set C [PTR]=A[NA]

B) [Update pointers] set PTR=PTR+1 and NB=NB+1

[End of if structure]

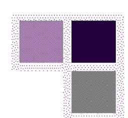
[End of loop]

3. [Assign remaining elements to C]

If NS>R, Then

Repeat for k=0, 1, 2,..., S-NB

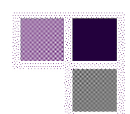
Set C [PTR+k] = B [NB+k]



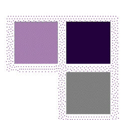
```
    [end of loop]
 Else
  Repeat for k=0, 1, 2, R-NA
   Set C [PTR+K] =A [NA+k]
    [end of loop]
[End of if structure]
4.  Exit
```

MERGING (A, R, B, S, C)

```
// let A and B be sorted array with R and S elements
NA=1; NB=1; PTR=1;
While (NA<=R &&NB<=S) {if (A [NA] <B [NB] {
C [PTR] =A [NA];
PTR=PTR+1;
NB=NB+1 ;}
}
If (NA>R){
For (k=0; k<=s-NB; ++k) {
C [PTR+k] =B [NB+k] ;}
Else
```



```
For (k=0; k<=R=NA; ++k) {
C [PTR+k] =A [NA+k] ;}}
```



مقدمة
introduction

غالبا ما تكون قيم المتغيرات في الحياة العملية غير مرتبة بالصورة التي يمكن للحاسب أن يتعامل معها . فمثلا كثيرا ما يقوم المرء بترتيب الأسماء هجائيا أو تصنيف الناس حسب أعمارهم أو أطوالهم من أجل أن يؤدي الحاسب دوره بأكفأ أسلوب . ويتطرق الفصل هذا إلى عدد من الطرق وأساليب الفرز نوردها كما هو مبين أدناه :

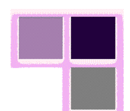
الفرز والترتيب التصاعدي الفقاعي Bubble sort program

مثال (٥٢) : الترتيب التصاعدي

يقوم البرنامج التالي بترتيب الأعداد ترتيبا تصاعديا : 6,5,3,9,77,78,66

```
/* Example (52) BUBBLE SORTING OF NUMBERS */
Main ()
{
Int n [8], m, i, j, z;
Printf ("input the number of values = ");
Scanf (" %d", &m);
Printf (" \n input the all values \n ");
For (i=1; i<m; ++i) Scanf (" %d", &n[i]);
For (i=1 ; i<m-1 ; ++i )
  For (j=i+1; j<m; ++j)
    If (n[i] >= n[j]) {
                        Z= n[i];
                       N[i] = n[j];
                       N[j] = z;
                                      }
Printf ("\n the values after bubble storing are \n") ;
For (i=0 ; i<m ; ++i)  printf ( " %d \n , n[j] ) ;
}
```

النتائج :



input the number of values = 8
input the all values
66 5 0 3 9 77 78 66
the values after bubble storing are
0
3
5
6
9
66
77
78

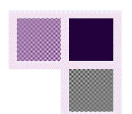
string Bubble sort program الفرز الفقاعي لأسماء رمزية

البرنامج (53) الفرز الفقاعي :

الفرق بين هذا البرنامج والبرنامج السابق هو اختلاف المعطيات ، فكانت هناك عددية ، وهنا رمزية ، والأسماء المراد ترتيبها هنا هي :

Ammar , Fatimah , omar , ahmad , jamal , saeed , yousef , mariam

```
/* Example (53) BUBBLE SORTING OF names */
#include <string.h>
main ()
{
Char *z ;
Int m , I , j ;
Char *st[ ]={"Ammar" , "Fatimah" , "omar" , "ahmad" , "jamal" , "saeed" , "yousef" ,
"mariam" };
M=8 ;
For (i=1 ; i<m-1 ; ++i )
  For (j=m-1 i<j ; --j )
    If ( strcmp ( st[j-1] , st [j] ) >0 )  {
                                            Z=st[j-1]
                                            St[j-1]=st[j];
                                            St[j]=z;
                                                          }
         For (i=0 ; i<m-1 ; ++i ) printf ("% s \n" , st[j] );
}
```



النتائج :

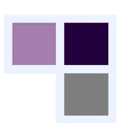
ahmad
Ammar
Fatimah
jamal
mariam
omar
saeed
yousef

مثال (54) : الفرز بالاختيار selection sorting

```
/* Example (54) selection sorting */
Main ()
{
Char *z ;
Char *name[ ]={"Ammar"
,"Fatimah","omar","ahmad","jamal","saeed","yousef","mariam"};
Int nmax =8;
Register int I , j , k ;
For (i=0 ; i<nmax-1 ; ++i )  {
  K=I;
  Z=name[i];
  For (j=I+1 ; j<nmax ; ++j) {
     If (strcmp (name[j] , z ) <0) {
        K=j;
        Z=name[j];  }
}
  Name[k]=name[i];
  Name [i]=z
}
For (i=0 ; i<nmax ; ++i) printf (" %s \n , name[i] );
}
```

النتائج :

ahmad
Ammar
Fatimah
jamal
mariam
omar
saeed
yousef

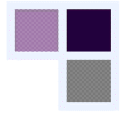


مثال (55) : الفرز بالادخال selection sorting

```
*\ program 55 : Sorting by tnsertion *\
Main()
{
Char *z ;
Char *name[ ]={"Ammar"
,"Fatimah","omar","ahmad","jamal","saeed","yousef","mariam"};
Int nmax =8;
Register int I , j;
For (i=1 ; i<nmax ; ++i )  {
 Z=name[i] ;
 J=i-1 ;
 While ( j>=0 && (strcmp ( z ,name[j] ) <0))  {
    Name[j+1]=name[j];
    j--;
   }
   Name[j+]=z;
 }
For (i=0 ; i<nmax ; ++i) printf (" %s \n , name[i] );
}
```

النـتـائـج :

```
ahmad
Ammar
Fatimah
 jamal
mariam
omar
saeed
yousef
```



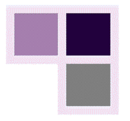
الفرز بطريقة شل (الفرز المتراكب أو التطبيقي) shellsort

مثال (56) : طريقة شل

```
\* program 56 : shell sort *\
Main ()
{
Char *name[ ]={"Ammar"
,"Fatimah","omar","ahmad","jamal","saeed","yousef","mariam"};
Int m[]={9,5,3,2,1};
Int nmax=8;
Register int a,b,c,t,v ;
Char *z;
For (v=0 ;v<5 ;++v)  {
  C=m[v];
  B=a-c;
  If (t==0)  {
    T=-c;
    T++ ;
    Name[t]=z ;
    }
  While((strcmp (z,name[b] ) <0)  && (b>0) && (b<nmax)) {
      Name[b+c]=name[b];
      b-=c;
      }
   Name[b+c]=z;
  }
}
For (a=0 ; a<8 ; ++a ) printf (" %s \n , name[a] );
}
```

النتائج :

```
ahmad
Ammar
Fatimah
 jamal
mariam
omar
saeed
yousef
```



مثال (57) : الفرز السريع quick sort

وهي أكفأ وأسرع الطرق الموجودة على الإطلاق .

```
/* program (57) : Quick sort */
Main ()
{
Int I;
Int data[]={80,50,40,30,15,35,70,90):
Int nmax=8;
Qstor (data , 0 , nmax-1);
For (i=0; i<nmax ; i++)  printf (" %s \n , data[i] );
}
/*quick sort function */
Qstor (number. Min , max )
Int number[];
Int min, max ;
{
Register int I ,j ;
Char a,b ;
i=min ;
J=max ;
A=number[( min+max)/2] ;
Do{
While (number[i]<a && i<max) i++
While (a<number[j] && j>min ) j--;
If (i<=j)  {
B=number[i] ;
Number[i]=number[j] ;
Number[j]=b ;
I++;
J++;
}
} while (i<=j);
If (min<j) qstor (number ,min ,j ) ;
If (i<max) qstor (number , I , max ) ;
}
```

النـتـائـج :

```
15
30
35
40
50
70
80
90
```

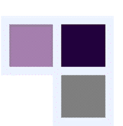
مثال (58) : الفرز السريع string quick sort

يختلف هذا البرنامج عن سابقه في أنه يرتب الكلمات ترتيبا قاموسيا (هجائيا)

```
/* program 58 : string quick sort */
Main ()
{
Int I, nmax=8
Char *name[ ]={"Ammar"
,"Fatimah","omar","ahmad","jamal","saeed","yousef","mariam"};
Qsorting (name ,0-1);
For (i=0 ;i<nmax ; nmax-1)  printf (" %s \n , name[a] );
}
qstoring (string ,min , max )
char *string[] ;
int min , max ;
{
Register int I ,j ;
Char*a , *b ;
I=min ;
J=max ;
A=string [ (min+max)/2 ];
Do{
While (strcmp (string[i],a ) <0 && i<max ) i++ ;
While (strcmp (string[i],a ) >0 && j<min ) j-- ;
If ( i<=j ) {
B=string[i] ;
String[i]=string[j] ;
String[j]=b ;
I++ ;
J-- ;
}
}while (i<=j ) ;
If ( min <j ) qstoring (string , min .j ) ;
If ( I <max ) qstoring (string , I , max )
}
```

النـتـائـج :

```
ahmad
Ammar
Fatimah
 jamal
mariam
omar
saeed
yousef
```



مثال (59) البحث التسلسلي عن عدد sequential search

يستعمل هذا البرنامج للبحث عن عدد معين من بين مجموعة أعداد ويمكن تحويله إلى برنامج للبحث عن كلمة من بين مجموعة من الكلمات بتعديل بسيط :

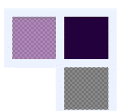
```
/*program 59 : sequential search */
Main ()
{
Int data []={7,4,5,6,3,2,10};
Int nmax=7 ;
Int key=3 ;
printf (" %s \n , sqsearch (data,nmax,key) );
}
 sqsearch (data,n,k) ;
int data ;
int n ;
int k ;
{
Register int I ;
For ( i=0 ; i<=n ; ++I )
If ( k==data[i] ) return ( i+1 ) ;
Return (-1) ;     /* no match exist */
}
```

النـتـائـج :

5

مثال (60) : البحث الثنائي binary search

```
/* program 60 : binary search */
Main ()
{
Int data []={3,4,5,6,7,9,10}
Int nmax=7 ;
Int key=7 ;
printf (" %s \n , bsearch (data,nmax,key) );
}
bsearch (data,n,k) ;
int data[] ;
int n ;
int k ;
{
Int min , max , mid ;
Min=0
```



طرق البحث searching method

ان البحث يتم عن عنصر معين ضمن مجموعة من العناصر وفي حالة وجود هذا العنصر ضمن مجموعة العناصر تحدد عملية البحث ويكون موقع العنصر ضمن هذه العناصر

١-البحث التابعي sequential method

هذه الطريقة من الطرق الأقل كفاءة والأكثر سهولة وتستخدم للبحث في السجلات عندما تكون مخزنة دون أي اعتبار للترتيب sorting

*وتتم هذه الطريقة كما يلي:-

١-اخذ كل سجل من سجلات الملف على حده بدأً بالسجل الأول.

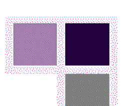
٢-تتم مقارنة مفاتيح البحث test key مع المفتاح الموجود في السجل الأول فإذا كشفت النتيجة عم وجود المفتاح في السجل يتوقف البحث ويعيد موقع هذا السجل.

٣- أما إذا لم يوافق مفتاح البحث مقابله في السجل فان عملية المقارنة تنتقل للسجل الذي يليه وهكذا حتى نهاية القائمة أو نجاح عملية المقارنة.

*في حالة عدم نجاح المقارنة مع نهاية القائمة فهذا يعني ان العنصر غير موجود

***فعالية البحث**

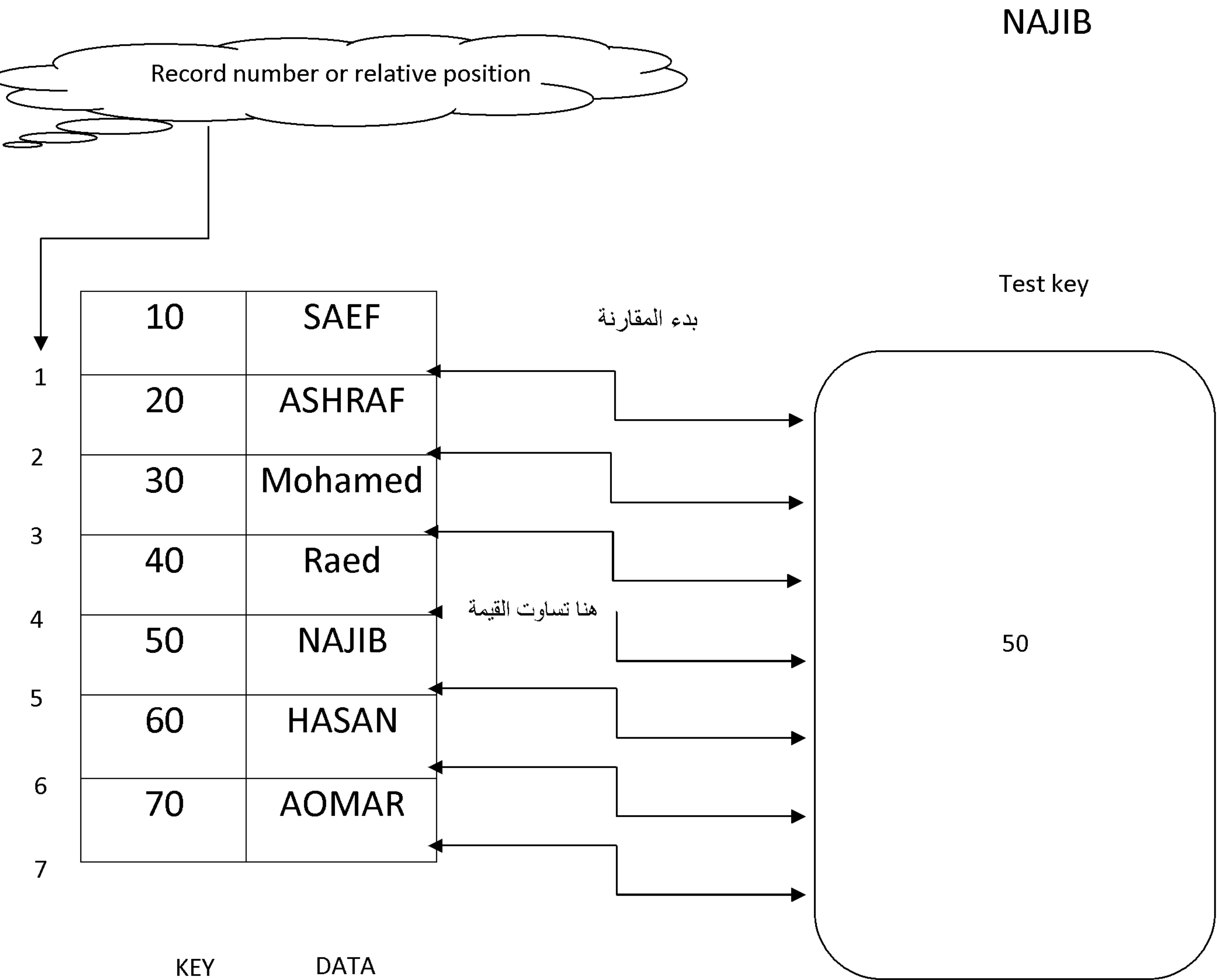
Efficiency=Rec/2



خوارزمية البحث التتابعي

Sequential search

افرض أن لدينا قائمة بالأسماء التالية ونريد استخدام طريقة البحث التتابعي عن اسم NAJIB



{Global declaration}

N=100 number of element to be sorted

Elements=record

Key:integer

Afile=array[0...n] of element

Procedure seqsearch(a:afile, i:integer , n , k:integer)

{

This procedure search a set of records,may be in a file, with key values

A[1].KEY, A[2].KEY , ...,A[N].KEY , FOR record such that A[i].KEY=KEY

If found , i is set to $\varnothing$

BEGINE

A[0].KEY=K

{dummy record to simplify the search

I=N

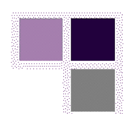
{

Which elements the need for end of file test}}

While A[I].KEY <>DO

I=I-1

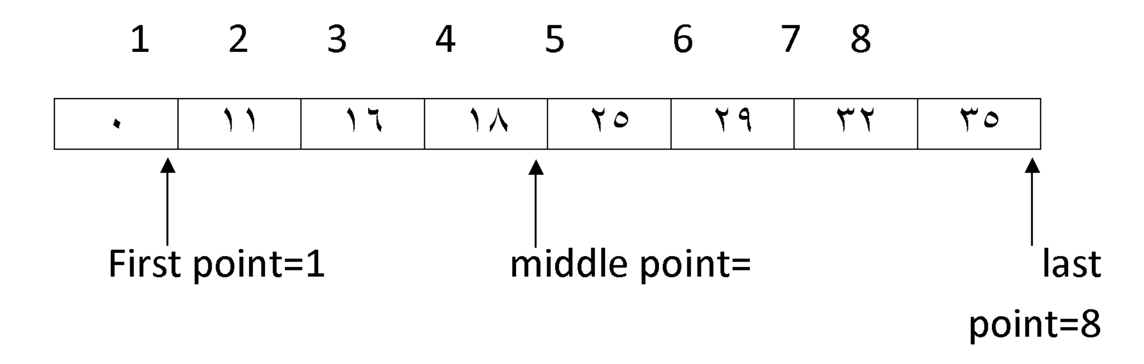
END{of sequential search procedure



٢-البحث الثنائي binary search

افرض انك أعطيت القائمة التالية من ٨ عناصر وهي مرتبه ترتيبا تصاعديا ونرغب بالبحث عن القيمة ٢٥ :-

١- نحدد قيمة مؤشر البداية (FIRST)وقيمة مؤشر النهاية (LAST)



(first+last)/2

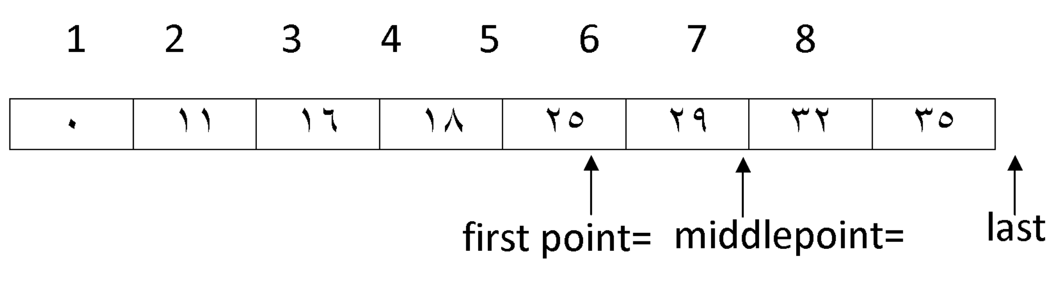
٣- نحدد نقطة الوسط حسب العلاقة:

Middle point= (first+last)/2=> (1+8)/2=4

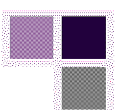
٤- نلاحظ أن قيمة Middle point=٤ هي١٨ وهي اصغر من القيمة اللتي نبحث عنها وهي ٢٥ ولذلك فان العناصر من (١ الى ٤) منطقيا لا تحتوي على العنصر ٢٥ المراد البحث عنه.

وبناء عليه فان العنصر مؤكدا بانه يقع في العناصر التالية والتي تقع بين

Middle point+١ والمؤشر last وبما ان الرقم الذي نبحث عنه ٢٥ اكبر من العنصر الوسط ١٨ فلذلك فاننا نقوم بتحريك مؤشر البداية first الى العنصر الذي يتبع الوسط باستخدام العلاقة first= Middle point+1 فتصبح القائمة كما يلي مع تعديل مؤشر المنتصف حسب المعطيات الجديدة



4+1=5 5+8)/2=6(



٥- نقارن العنصر الواقع في وسط القائمة الجديدة (وهو العنصر السادس) مع القيمة التي نبحث عنها:-

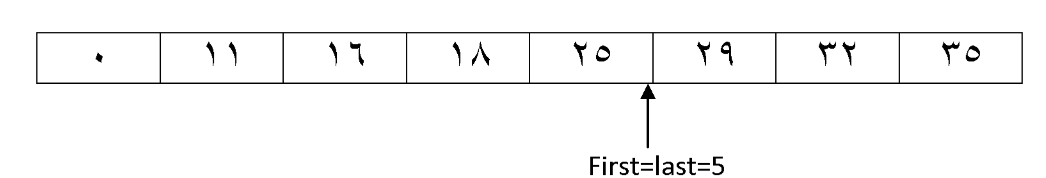
فنجد ان قيمة العنصر الوسط middle point=6 والذي محتواه ٢٩ مع ٢٥ فنجد ان ٢٩>٢٥ وبناءً عليه نقوم بتغيير قيمة last الى العنصر الذي ياتي قبل الوسط مباشرة وذلك باستخدام العلاقة

Last=middle point=-1 => 6-1=5

وبذلك تصبح قيمتي first=last=5

1 2 3 4 5 6 7

8



٠	١١	١٦	١٨	٢٥	٢٩	٣٢	٣٥

٦- نقوم بحساب الوسط الجديد middlepoint=(5+5)/2=5

ثم نقارن العنصر في الوسط الجديد (الموقع الخامس) مع الرقم ٢٥ فنلاحظ انه يساويه وبالتالي فان موقع العنصر ٢٥ الذي نبحث عنه هو الخامس.

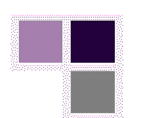
ملاحظة/

اما اذا كان العنصر غير موجود في القائمة فإنه ينبغي ايقاف البحث عندما تصبح قيمة first اكبر من قيمة last.

EFFICINCAY=(Log$_2$number Rec)+1

للمثال اعلاه log$_2$ 8 +1=3+1

أي نحتاج لقائمة من ٨ عناصر لأربع مقارنات كحد أعلى.



```
PROCEDURE BIN SEARCH (KEY:INTEGER;VAR:IIM:DATA
ARRAY;VARFOUND:BOOLEAN)

First=1;

Last=actual

REPEAT

MIDDLEPOINT= (FIRST+LAST)/2

IF KEY>info [middlepoint].key then

First=middlepoint+1

Else if key< info [middlepoint].key then

Else {key found}

  Item=inf [middlepoint].data

Found=true

End

Until found or(last<first)

End{binary search}
```

برنامج البحث التتابعي

```
#include<iostream.h>

Int linearsearch(cost int[], int , int);

Int main()

{

Cost int arraysize=100;

Int a[arraysize] , searchkey , element;

For(int x=0 ; x<arraysize ; x++)

A[x]=2*x;

Cout<<"enter unteger search key"

Cin>>searchkey;

Element=linearserch(a , searchkey , arraysize);

If (element != -1)

Cout<<"found value in element"<<element;

Else

Cout<<"value not vound"<<endl;

Return 0;

}

Int linear serch (const int array[] , int key,Int sizeofarry)

{

For(int v=0 ; v<sizeofarray ; v++)

If(array[n]==key)

Return v; }
```

برنامج البحث الثنائى

```
//binary search

Int binarysearch (int b[] , int searchkey , int low, int high , int
size)

{

Int middle;

While (low<=high)

{

Middle=(low+high)/2;

If(searchkey==b[middle])

Return middle;

Else if(searchkey < b [middle])

High=middle -1;

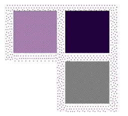
Slse low=middle +1;

}

Return -1

//searchkey not found

}
```



الهياكل الديناميكية :-

قبل الحديث عن المكدسات stacks ، القوائم lists ، والطوابير queues فأنه من الضروري الحديث عن مجموعة الهياكل الديناميكية والتي تستخدم في كتابة البرامج التي تخدم هذه التراكيب .

المؤشرات Pointers :-

تعتبر المؤشرات من الخصائص المهمة التي تمتاز بها لغة C++ والتي تعتبر من أقوى اللغات لوصف تراكيب البيانات .

لفهم المؤشرات لابد من تصور ما تمثله على مستوى الأله حيث تقسم الذاكرة في الحاسب إلى Bytes قادرة على تخزين 8-bits من البيانات ولتسهيل عملية الوصول إلى البايت

المطلوب فأن كل بايت يعطى عنوانا address فريدا (أي غير متكرر) ، فلو افترضنا أن عدد البايتات هو n فان العناوين المطلوبة هي من 0 إلى n-1 .

بالتالي فانه عند عملية الاحتفاظ بعنوان متغير معين في متغير خاص يحمل رقم الموقع location . فعندها يطلق على ذالك المتغير الخاص أسم مؤشر pointer .

فمثلا لو احتفظنا بعنوان المتغير i = 10101111 في المتغير p=2 حيث القيمة ٢ تشير إلى موقع i بالتالي فان p سوف يشير إلى المتغير i كما يلي

$$P = 2 \rightarrow i = 10101111$$

بحيث i تمثل قيمة المتغير في الذاكرة و p تمثل رقم موقع المتغير في الذاكرة .

يتم الإعلان عن المؤشر في لغة C++ كما يلي

```
Int *p ;
```

وفي هذه الحالة يقال أن المتغير p هو مؤشر يشير إلى متغير من النوع الصحيح

هذا ويمكن استخدام المؤشرات للإشارة إلى أنوع مختلفة من المتغيرات مثل

```
Int *p ; // pointer to integers

float *q ; // pointer to float

char *r  ; // pointer to characters
```

معامل العنوان (&) ومعامل المحتوى (*) :-

مع المؤشرات يتم استخدام أداتان مهمتان في لغة C++ وهي الأداة & والتي تستخدم لإعطاء عنوان متغير في الذاكرة فعلى سبيل المثال

```
Mem = &x  ;
```

فإذا كان المتغير x واقعا في الموقع 1000 من الذاكرة وكانت قيمة x في الموقع هي 3 فان جملة التعين السابقة تعطي المتغير mem في موقعه القيمه 1000 وهي عنوان x في الذاكرة وعليه فان معنى الجملة هو أعطي mem في موقعه عنوان x .

```
mem = &x = 1000
```

أما الأداة * فإنها تستخدم لإعطاء قيمة المتغير المؤشر(المشار اليه اي قيمة x)

وبالتالي نكتب

```
mem = & x

Y = * mem  ;                    .........y=3
```

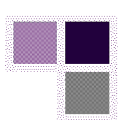
بالتالي الإعلان في البرنامج سيكون كتالي

```
Int  x , * mem  ;

mem= &x  ;
```

مثال اكتب برنامج يطبع قيمتي a[1] , a[0] باستخدام المؤشرات

```
#include< iostream.h>
main()
 {
int  a[2] , * p1 , * p2  ;
P1= &a ;
P2= p1 +2
cout<<  * p1 <<  * p2  ;
}
```



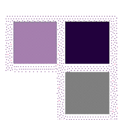
التراكيب structures

التركيب هو عبارة عن مجموعه من العناصر التي تربطها مع بعضها البعض علاقات وثيقة وهذه العناصر من الممكن إن تكون من أنواع مختلفة المعطيات فمثلا

```
struct  {
      int  i      ;
      float  f  ;
      char  z   ;
 }  s   ;
```

الشكل العام للتركيب هو كما يلي

```
struct    type {
   var 1   ;
    var 2  ;
        .
        .
        .
}  structure name    ;
```



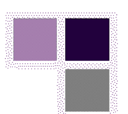
اكتب برنامج يصف الطقس باستخدام التركيب .

```
#include< iostream.h>
struct  weather   {
  float  temp    ;
  float  press     ;
 }  climate   ;


Main ( )
{
Weather   climate    ;
Climate  .  temp = 25.3   ;
Climate  .  prees = 755    ;
Cout<< " \ n  temp= " << climate . temp   ;
Cout << " \ n press= " <<  climate  . press   ;
}
```

لغة ++c تسمح باستخدام مصفوفة التراكيب structure array على سبيل المثال

```
#include<iostream.h>
#defin  m   30
Struct   address {
  Char   name[20]   ;
   Char   street[30]   ;
    Int  pox[8]           ;
  }  information         ;
main( ) {
     address    information[m]  ;
    for( i=0 ; i=30 ; i++)
cin>> information[i] . name  ;
cin>> information[i]  . street ;
cin>> information[i] . pox         ;
}
```



الاصناف classes

يعد الصنف في c++ من أهم ميزات هذه اللغة في معالجة البيانات .والصيغة العامة للصنف هي كما يلي :-

```
class      class name  {

closed function   and  var. of  class

public :

        open    function   and  var. of class

} list  of  objects      ;
```

على سبيل المثال البرنامج التالي :-

```
#include<iostream.h>
class   myclass {
      int   a ;
   public :
         void    set_a( int  num )    ;
          int       get_a()     ;
   }   ;
Void   myclass :: set_a( int   num )
    {   a=num     ;     }
int       myclass :: get_a( )
   {   return    a    ; }
main()
{
Myclass    ob1  ,  ob2   ;
ob1 . set_a( 10 )   ;
ob2 . set_a( 99 )   ;
cout<<  ob1 . get_a( ) << " \ n "    ;
cout<< ob2  . get_a( ) << " \n "
return   0
```

تمارين على الهياكل الديناميكية

Classes are generally declared using the keyword `class`, with the following format:

```
class class_name {
  access_specifier_1:
    member1;
  access_specifier_2:
    member2;
  ...
  } object_names;
```

By default, all members of a class declared with the `class` keyword have private access for all its members. Therefore, any member that is declared before one other class specifier automatically has private access. For example:

```
class CRectangle {
    int x, y;
  public:
    void set_values (int,int);
    int area (void);
  } rect;
```

The only members of rect that we cannot access from the body of our program outside the class are x and y, since they have private access and they can only be referred from within other members of that same class.

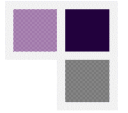
Here is the complete example of class CRectangle:

```
// classes example
#include <iostream>
using namespace std;

class CRectangle {
    int x, y;
  public:
    void set_values (int,int);
    int area () {return (x*y);}
};

void CRectangle::set_values (int a, int b) {
  x = a;
  y = b;
}
```

```
area: 12
```



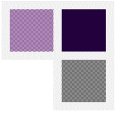
```
int main () {
  CRectangle rect;
  rect.set_values (3,4);
  cout << "area: " << rect.area();
  return 0;
}
```

One of the greater advantages of a class is that, as any other type, we can declare several objects of it. For example, following with the previous example of class `CRectangle`, we could have declared the object `rectb` in addition to the object `rect`:

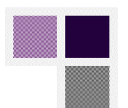
```
// example: one class, two objects
#include <iostream>
using namespace std;

class CRectangle {
    int x, y;
  public:
    void set_values (int,int);
    int area () {return (x*y);}
};
```

```
rect area: 12
rectb area: 30
```



```
void
CRectangle::set_values
(int a, int b) {
  x = a;
  y = b;
}
int main () {
  CRectangle rect,
rectb;
  rect.set_values (3,4);
  rectb.set_values
(5,6);
  cout << "rect area: "
<< rect.area() << endl;
  cout << "rectb area: "
<< rectb.area() << endl;
  return 0;
}
```



In order to avoid that, a class can include a special function called `constructor`, which is automatically called whenever a new object of this class is created. This constructor function must have the same name as the class, and cannot have any return type; not even `void`.

We are going to implement `CRectangle` including a constructor:

```
// example: class constructor

#include <iostream>

using namespace std;

class CRectangle {

    int width, height;

  public:

    CRectangle (int,int);

    int area () {return (width*height);}

};

CRectangle::CRectangle (int a, int b) {

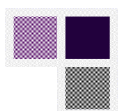
  width = a;

  height = b;

}
```

```
rect area: 12

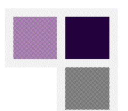
rectb area: 30
```



<pre>int main () { CRectangle rect (3,4); CRectangle rectb (5,6); cout << "rect area: " << rect.area() << endl; cout << "rectb area: " << rectb.area() << endl; return 0; }</pre>	
<pre>// example on constructors and destructors #include <iostream> using namespace std; class CRectangle { int *width, *height; public: CRectangle (int,int); ~CRectangle (); int area () {return (*width * *height);}</pre>	<pre>rect area: 12 rectb area: 30</pre>

```
};

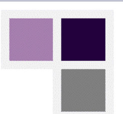

CRectangle::CRectangle
(int a, int b) {
  width = new int;
  height = new int;
  *width = a;
  *height = b;
}
CRectangle::~CRectangle
() {
  delete width;
  delete height;
}
int main () {
  CRectangle rect (3,4),
rectb (5,6);
  cout << "rect area: "
<< rect.area() << endl;
  cout << "rectb area: "
<< rectb.area() << endl;
  return 0;
}
```



```
// overloading class
constructors
#include <iostream>
using namespace std;


class CRectangle {
    int width, height;
  public:
    CRectangle ();
    CRectangle
(int,int);
    int area (void)
{return (width*height);}
};
CRectangle::CRectangle
() {
  width = 5;
  height = 5;
}
CRectangle::CRectangle
(int a, int b) {
  width = a;
  height = b;
}
```

```
rect area: 12
rectb area: 25
```

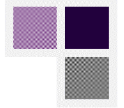


```
int main () {
  CRectangle rect (3,4);
  CRectangle rectb;
  cout << "rect area: "
<< rect.area() << endl;
  cout << "rectb area: "
<< rectb.area() << endl;
  return 0;
}
```

In this case, `rectb` was declared without any arguments, so it has been initialized with the constructor that has no parameters, which initializes both `width` and `height` with a value of 5.

Important: Notice how if we declare a new object and we want to use its default constructor (the one without parameters), we do not include parentheses `()`:

```
CRectangle rectb;   // right
CRectangle rectb(); // wrong!
```



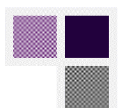
```
// pointer to classes
example
#include <iostream>
using namespace std;

class CRectangle {
    int width, height;
  public:
    void set_values
(int, int);
    int area (void)
{return (width *
height);}
};

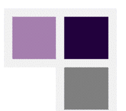
void
CRectangle::set_values
(int a, int b) {
  width = a;
  height = b;
}

int main () {
```

```
a area: 2
*b area: 12
*c area: 2
d[0] area: 30
d[1] area: 56
```



```
  CRectangle a, *b, *c;
  CRectangle * d = new
CRectangle[2];
  b= new CRectangle;
  c= &a;
  a.set_values (1,2);
  b->set_values (3,4);
  d->set_values (5,6);
  d[1].set_values (7,8);
  cout << "a area: " <<
a.area() << endl;
  cout << "*b area: " <<
b->area() << endl;
  cout << "*c area: " <<
c->area() << endl;
  cout << "d[0] area: "
<< d[0].area() << endl;
  cout << "d[1] area: "
<< d[1].area() << endl;
  delete[] d;
  delete b;
  return 0;
}
```



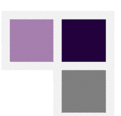
Next you have a summary on how can you read some pointer and class operators (`*`, `&`, `.`, `->`, `[ ]`) that appear in the previous example:

expression	can be read as
*x	pointed by x
&x	address of x
x.y	member y of object x
x->y	member y of object pointed by x
(*x).y	member y of object pointed by x (equivalent to the previous one)
x[0]	first object pointed by x
x[1]	second object pointed by x
x[n]	(n+1)th object pointed by x

```
// vectors: overloading operators example
#include <iostream>
using namespace std;


class CVector {
  public:
    int x,y;
    CVector () {};
```

```
4,3
```

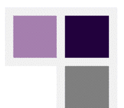


```
    CVector (int,int);
    CVector operator +
(CVector);
};


CVector::CVector (int a,
int b) {
  x = a;
  y = b;
}


CVector
CVector::operator+
(CVector param) {
  CVector temp;
  temp.x = x + param.x;
  temp.y = y + param.y;
  return (temp);
}


int main () {
  CVector a (3,1);
  CVector b (1,2);
  CVector c;
```



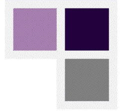
```
  c = a + b;
  cout << c.x << "," <<
c.y;
  return 0;
}
```

It may be a little confusing to see so many times the `CVector` identifier. But, consider that some of them refer to the class name (type) `CVector` and some others are functions with that name (constructors must have the same name as the class). Do not confuse them:

```
CVector (int, int);          // function name
CVector (constructor)
CVector operator+ (CVector);   // function
returns a CVector
```

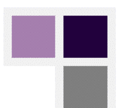
The function `operator+` of class `CVector` is the one that is in charge of overloading the addition operator (`+`). This function can be called either implicitly using the operator, or explicitly using the function name:

```
c = a + b;
c = a.operator+ (b);
```



```
// this
#include <iostream>
using namespace std;
class CDummy {
  public:
    int isitme (CDummy& param);
};
int CDummy::isitme (CDummy& param)
{
  if (&param == this) return true;
  else return false;
}
int main () {
  CDummy a;
  CDummy* b = &a;
  if ( b->isitme(a) )
    cout << "yes, &a is b";
  return 0;
}
```

```
yes, &a is b
```



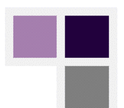
It is also frequently used in `operator=` member functions that return objects by reference (avoiding the use of temporary objects). Following with the vector's examples seen before we could have written an `operator=` function similar to this one:

```
CVector& CVector::operator= (const CVector& param)
{
  x=param.x;
  y=param.y;
  return *this;
}
```

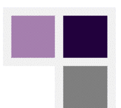
```
// static members in classes
#include <iostream>
using namespace std;

class CDummy {
  public:
    static int n;
    CDummy () { n++; };
    ~CDummy () { n--; };
};
```

```
7
6
```



int CDummy::n=0; int main () { CDummy a; CDummy b[5]; CDummy * c = new CDummy; cout << a.n << endl; delete c; cout << CDummy::n << endl; return 0; }	
// friend functions #include <iostream> using namespace std; class CRectangle { int width, height; public: void set_values (int, int); int area () {return (width * height);} friend CRectangle duplicate (CRectangle);	24



```
};

void
CRectangle::set_values
(int a, int b) {

  width = a;

  height = b;

}

CRectangle duplicate
(CRectangle rectparam)

{

  CRectangle rectres;

  rectres.width =
rectparam.width*2;

  rectres.height =
rectparam.height*2;

  return (rectres);

}

int main () {

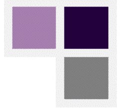
  CRectangle rect,
rectb;

  rect.set_values (2,3);

  rectb = duplicate
(rect);

  cout << rectb.area();

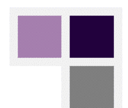
  return 0;
```



``}```	

```
// derived classes
#include <iostream>
using namespace std;
class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b;}
  };
class CRectangle: public CPolygon {
  public:
    int area ()
      { return (width * height); }
  };
class CTriangle: public CPolygon {
  public:
```

```
20
10
```

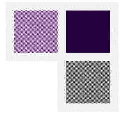


```
    int area ()
      { return (width * height / 2); }
  };
int main () {
  CRectangle rect;
  CTriangle trgl;
  rect.set_values (4,5);
  trgl.set_values (4,5);
  cout << rect.area() << endl;
  cout << trgl.area() << endl;
  return 0;
}
```

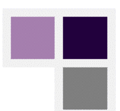
```
// constructors and derived classes
#include <iostream>
using namespace std;
class mother {
  public:
    mother ()
```

```
mother: no parameters
daughter: int parameter

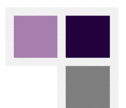
mother: int parameter
son: int parameter
```



```
      { cout << "mother: 
no parameters\n"; }
    mother (int a)
      { cout << "mother: 
int parameter\n"; }
};
class daughter : public 
mother {
  public:
    daughter (int a)
      { cout << 
"daughter: int 
parameter\n\n"; }
};
class son : public 
mother {
  public:
    son (int a) : mother 
(a)
      { cout << "son: 
int parameter\n\n"; }
};
int main () {
  daughter cynthia (0);
  son daniel(0);
  return 0;
```

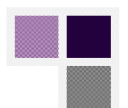


Code	Output
}	
// multiple inheritance #include <iostream> using namespace std; class CPolygon { protected: int width, height; public: void set_values (int a, int b) { width=a; height=b;} }; class COutput { public: void output (int i); }; void COutput::output (int i) { cout << i << endl; } class CRectangle: public CPolygon, public COutput	20 10



```
{
  public:
    int area ()
      { return (width *
height); }
  };
class CTriangle: public
CPolygon, public COutput
{
  public:
    int area ()
      { return (width *
height / 2); }
  };
int main () {
  CRectangle rect;
  CTriangle trgl;
  rect.set_values (4,5);
  trgl.set_values (4,5);
  rect.output
(rect.area());
  trgl.output
(trgl.area());
  return 0;
}
```

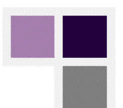
// pointers to base class #include <iostream> using namespace std; class CPolygon { protected: int width, height; public: void set_values (int a, int b) { width=a; height=b; } }; class CRectangle: public CPolygon { public: int area () { return (width * height); } }; class CTriangle: public CPolygon { public:	20 10



```
    int area ()
      { return (width * height / 2); }
  };
int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << rect.area() << endl;
  cout << trgl.area() << endl;
  return 0;
}
```

```
// virtual members
#include <iostream>
using namespace std;
class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int
a, int b)
      { width=a;
height=b; }
    virtual int area ()
      { return (0); }
  };
class CRectangle: public
CPolygon {
  public:
    int area ()
      { return (width *
height); }
  };
class CTriangle: public
CPolygon {
  public:
    int area ()
```

```
20
10
0
```



```
      { return (width * 
height / 2); }

  };

int main () {

  CRectangle rect;

  CTriangle trgl;

  CPolygon poly;

  CPolygon * ppoly1 = 
&rect;

  CPolygon * ppoly2 = 
&trgl;

  CPolygon * ppoly3 = 
&poly;

  ppoly1->set_values 
(4,5);

  ppoly2->set_values 
(4,5);

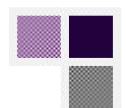
  ppoly3->set_values 
(4,5);

  cout << ppoly1->area() 
<< endl;

  cout << ppoly2->area() 
<< endl;

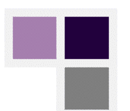
  cout << ppoly3->area() 
<< endl;

  return 0;
```

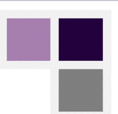


}	
// abstract base class #include <iostream> using namespace std; class CPolygon { protected: int width, height; public: void set_values (int a, int b) { width=a; height=b; } virtual int area (void) =0; }; class CRectangle: public CPolygon { public: int area (void) { return (width * height); } }; class CTriangle: public CPolygon { public:	20 10

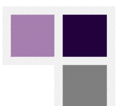
```
    int area (void)
      { return (width * 
height / 2); }
  };
int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon * ppoly1 = 
&rect;
  CPolygon * ppoly2 = 
&trgl;
  ppoly1->set_values 
(4,5);
  ppoly2->set_values 
(4,5);
  cout << ppoly1->area() 
<< endl;
  cout << ppoly2->area() 
<< endl;
  return 0;
}
```



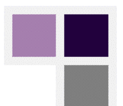
// pure virtual members can be called // from the abstract base class #include <iostream> using namespace std; class CPolygon { protected: int width, height; public: void set_values (int a, int b) { width=a; height=b; } virtual int area (void) =0; void printarea (void) { cout << this->area() << endl; } }; class CRectangle: public CPolygon { public:	20 10



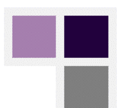
```
    int area (void)
      { return (width *
height); }
  };
class CTriangle: public
CPolygon {
  public:
    int area (void)
      { return (width *
height / 2); }
  };
int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon * ppoly1 =
&rect;
  CPolygon * ppoly2 =
&trgl;
  ppoly1->set_values
(4,5);
  ppoly2->set_values
(4,5);
  ppoly1->printarea();
  ppoly2->printarea();
  return 0;
```



}	
// dynamic allocation and polymorphism #include <iostream> using namespace std; class CPolygon { protected: int width, height; public: void set_values (int a, int b) { width=a; height=b; } virtual int area (void) =0; void printarea (void) { cout << this->area() << endl; } }; class CRectangle: public CPolygon { public: int area (void) { return (width * height); }	20 10



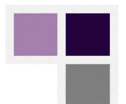
```
  };
class CTriangle: public
CPolygon {
  public:
    int area (void)
      { return (width *
height / 2); }
  };
int main () {
  CPolygon * ppoly1 =
new CRectangle;
  CPolygon * ppoly2 =
new CTriangle;
  ppoly1->set_values
(4,5);
  ppoly2->set_values
(4,5);
  ppoly1->printarea();
  ppoly2->printarea();
  delete ppoly1;
  delete ppoly2;
  return 0;
}
```



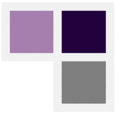
Arrays

Code	Output
`// arrays example` `#include <iostream>` `using namespace std;` `int billy [] = {16, 2, 77, 40, 12071};` `int n, result=0;` `int main ()` `{` `  for ( n=0 ; n<5 ; n++ )` `  {` `    result += billy[n];` `  }` `  cout << result;` `  return 0;` `}`	12206
`// arrays as parameters` `#include <iostream>` `using namespace std;` `void printarray (int arg[], int length) {` `for (int n=0; n<length;n++)`	5 10 15 2 4 6 8 10

<pre> cout << arg[n] << " "; cout << "\n"; } int main () { int firstarray[] = {5, 10, 15}; int secondarray[] = {2, 4, 6, 8, 10}; printarray (firstarray,3); printarray (secondarray,5); return 0; }</pre>	
<pre>// null-terminated sequences of characters #include <iostream> using namespace std; int main () { char question[] = "Please, enter your first name: "; char greeting[] = "Hello, "; char yourname [80]; cout << question;</pre>	Please, enter your first name: John Hello, John!



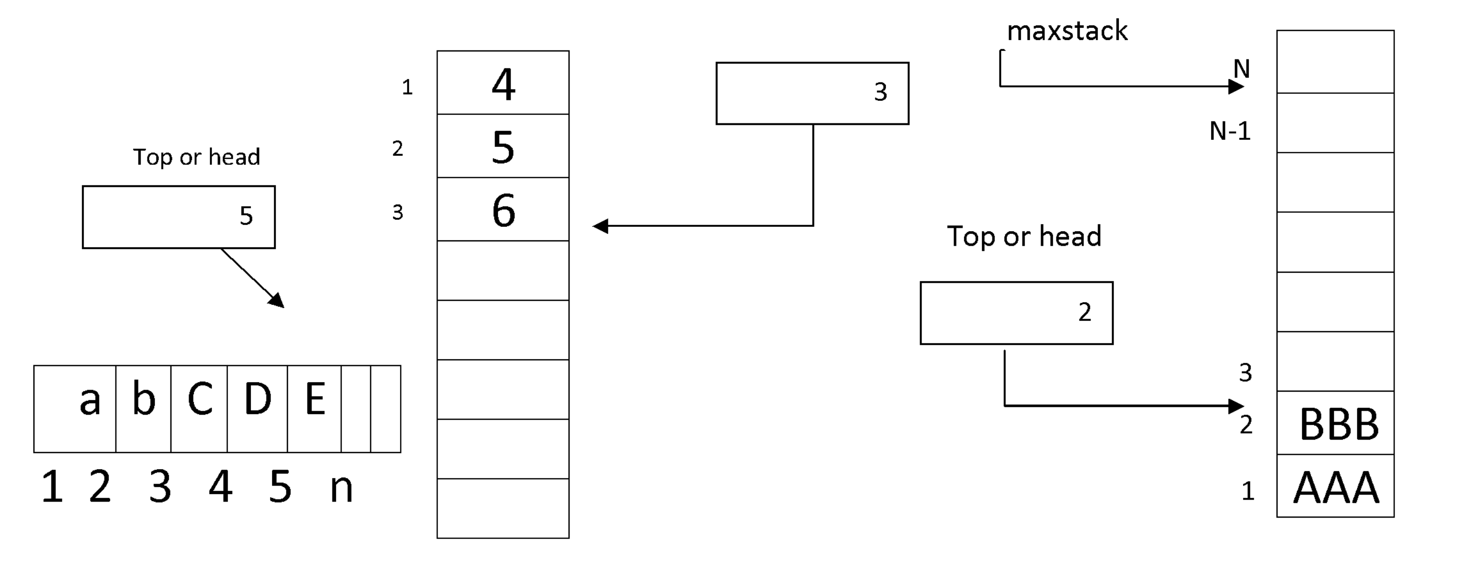
```
  cin >> yourname;
  cout << greeting <<
yourname << "!";
  return 0;
}
```



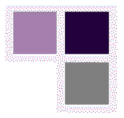
الحزم

الحزمة(stack):-

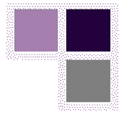
يمكن تصور الحزمة على أنها مجموعة من المواقع المستخدمة لتخزين البيانات وحيث تتم عملية الإضافة والحذف من طرف واحد(البداية). وعلية ينطبق على الحزمة مبدء (last in first out LIFO)




```
Const int maxstack=10;//small value for lesting
Class stack{
          Public:
         Stack();
          Bool empty()const;
          Error_code pop();
         Error_code top (stack_entry&item) const;
         Error_code push (const stack_entry & item);
        Private:
        Int count;
       Stack_entry entry[maxstack];
};
```



```
Error_code stack::push (const stack_entry & item)
//if the stack is not full,item is added to the top of the stack.if the stack
is full,an error_code of overflow is returned and the stack is left
unchanged
{
Error_code outcome=success;
If  (count >=maxstack)
Outcome=overflow;
Else
Entry[count ++]=item;
Return outcome;
}
Error_code stack::pop()
// if the stack is not empty, the top of the stack.if the stack is removed,if
the stack is empty ,an error_code of underflow is returned
{
Error_code outcome=success;
If  (count ==0)
Outcome= underflow;
Else
--count;
Return outcome;
}
```



```
Error_code stack(stack_entry & item) const

//in the stack is not empty ,the top of the stackis returned in item.if the stack is empty an error_code of underflow is returned

{

Erorre_code outcome=success;

If(count ==0)

Outcome=entry[count-1];

Return outcome;

}

Bool stack::empty()const

//if stack is empty , true is returned .otherwise false is returned

{

Bool outcome=true;

If(count>0)outcome=false;

Returned outcome;

}
```

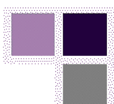
*the other method method of our stack is the constructure . the purpose of the constructor is to initialize any new stack object as empty.

```
Stack ::stack()

//the stack is initialize to be empty

{

Count=0;  }
```



برنامج بسيط للمكدس بلغة c++

```
#include<stack>

int main()

{

int n;

double  item;

stack<double>numbers;//declare and initialize a stack of numbers

cout<<"type in integer n followed by decimal numbers"<<"the numbers will be printed in reverse order "<<endl ;

cin>>n;

for(int i-0 , i<n ,i++)

{

Cin>>item;

Numbers.push(item);

}

Cout<<endl;

While(!numbers.empty() )

{

Cout <<numbers.top()<<"";

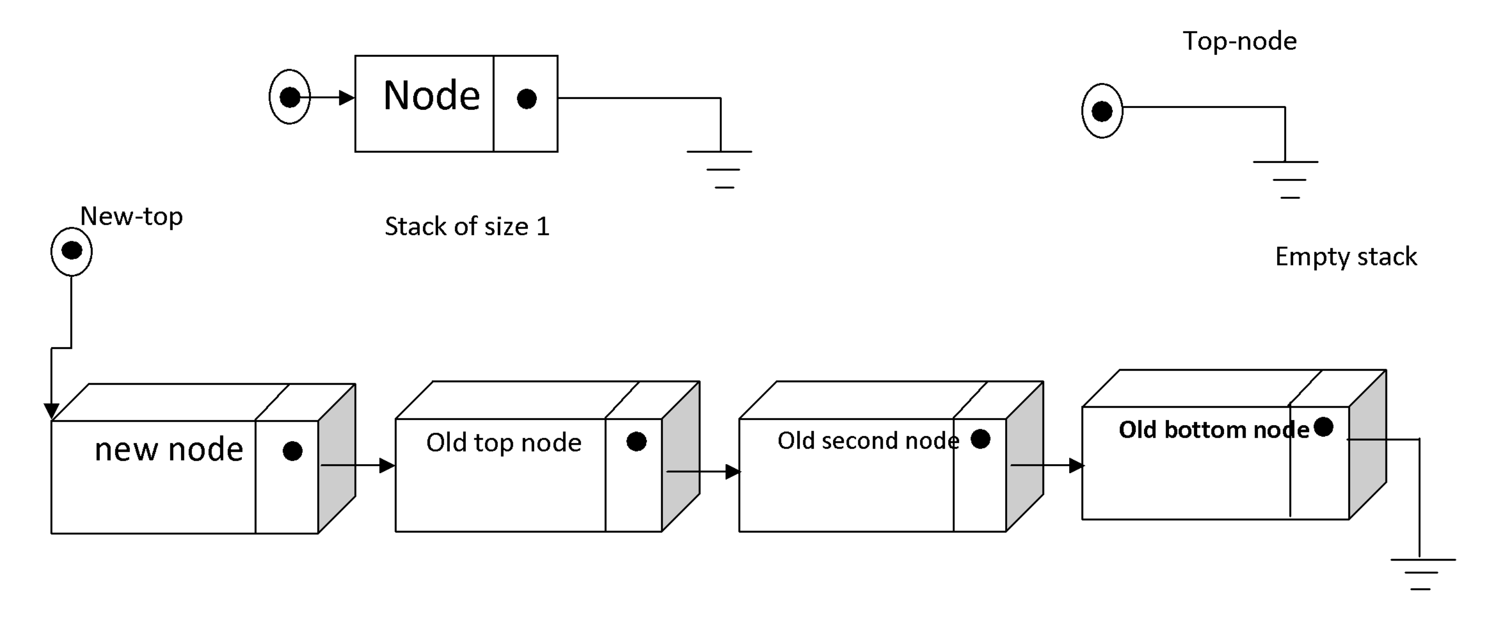
Numbers.pop();

}

Cout<<endl;

}
```

Linked Stacks




```
Error-code stack::push (const stack-entry & item)

//stack-entry itemis added to top of stack ; returns success or returns a
code of overflow if dynamic memory is exhausted

{

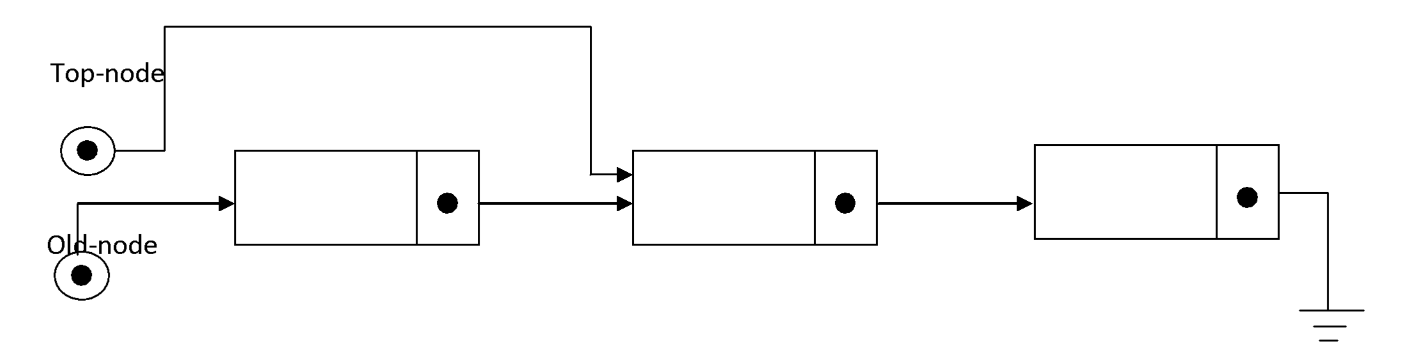
Node*new-top=new node(item,top-node)

If(new-top==null)return overflow;

Top-nod =new-top;

Return success;

}
```



Error-code stack::pop()

//the top of the stack is removed . if the stack is empty the method return underflow;

Otherwise is return success.

{

Node*old-top= top-node;

If(top-node==null)return underflow;

Top-nod =old-top->next;

Delete old-top;

Return success;

}

بعض عمليات المكدس في ال c++ بالإضافة الى pop و push :-

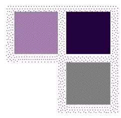
Error_code stack::(stack-entry&item)const;

The top of a npnempty stack is copied to iteam,a code of fail is returned if the stack is empty.

Bool stack ::empty()const();

A result of true is returned if the stack is empty;

Otherwise false is returned;



استخدام المكدسات

أولاً/إيجاد قيمة التعبير الرياضي باستخدام المكدس/

Infix notation الوسطى	التبعي أو الجدي postfix notation	Prefix notation القبلي
5+9	5 9 +	+5 9
A/B	A B /	/ A B
A + B –C	A B + C -	- + A B C
A*(B+C)	A B C + *	*A+BC
A+B/C	ABC/+	+A/BC
(A+B)/C	AB+C/	/+ABC
SINX/X	SINX X /	/ SINX X

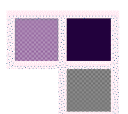
- طرق كتابة التعبير الحسابية:-

أولويات العمليات:

١- الضرب والقسمة (*،/).

٢-الجمع والطرح (+،-).

وفي حالة تساوي الأولويات فان القاعدة المستخدمة هي التزام السياق الذي وردت فيه من اليسار إلى اليمين وذلك ما لم تكن هناك أقواس.



الطريقة الأولى/

التحويل من النظام الوسطى إلى البعدي

(Converting infix to postfix)

استخدام المكدس لتمثيل التعابير الحسابية بصيغة prnor postfix...

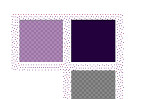
من الجدير ذكره ان هذه الطريقة هي ما تستخدمها المترجمات في ترجمة التعابير الحسابية و تهيئتها للتنفيذ اللاحق.

بمعنى آخر بواسطة هذه الطريقة يتم تحويل أي تعبير حسابي من الصيغة (البينية) infix notation إلى الشكل

postfix notation (البعدي).

- الخطوات الخوارزمية الأساسية للتحويل من البيني إلى الالبعدي:-

١- تهيئة مكدسين احدهما للترميز البولياني postfix والأخر للعوامل الحسابية the operation.
٢- إقراء التعبير الحسابي من اليسار لليمين رمزا بعد الآخر.
٣- إذا كان الرمز عددا أو متغيرا ندفعه push إلى مكدس الترميز.
٤- إذا كان الرمز معامل حسابي ندفعه push إلى مكدس العمليات.
٥- فرغ مكدس العمليات الحسابية من العوامل الحسابية وادفعها إلى مكدس التدوين البولياني المعكوس حتى تصل إلى النهاية المسدودة للمكدس أو عندما يكون العامل في القيمة الأقل أفضلية من العامل الذي بين يديك أدفعة إلى قمة مكدس التدوين البولياني.
٦- عند انتهاء مسح التعبير الحسابي المعطى فرغ pop مكدس العمليات من عوامله وادفع push كل منها في مكدس الترميز البولياني.



الطريقة الثانية :

في هذه الطريقة يستخدم مكدسين..

أ- مكدس العوامل الحسابية operators

ب- مكدس الأعداد والمتغيرات operands stack

ويتم احتساب قيمة تعبير في صيغة Infix notation

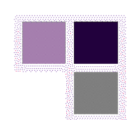
تسلسل التنفيذ	operands stack	operators stack	Infix notation صيغة بينية
1			3+7*2
2	3		+7*2
3	3	+	7*2
4	37	+	*2
5	37	+*	2
6	372	+*	
7	314	+	
8	17		
9	17	-	
10	176	-	
11	11		

أوجد قيمة التعبير 7*8-6*3\2

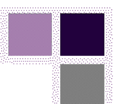
(أكتب برنامج) ————

خوارزمية التحويل من الصيغة infix الى postfix

```
Procedure convert (infix:CHAR; VAR POSTFiX: CHAR ,
M:integer , VAR P: integer)
S: stack; J: integer; Element: stack Data;
Stack create(s); p=0;
For J=1 To n
If INFIX [j] IN ['*' , '\' , '+' , '-'] Then
   If not stack Empty(s) Then
   While (PRIOITY(s.Data[s.Top]>= PRIOITY
   (infix[j]))AND(s.TOP=0) AND
   (s.Data[S.Top]= '(' )Do
   Pop(s,ELEMENT);
  P=p+1;
  Postfix[p]= ELEMENT;
  End;
Push(s,INFIX[J]);
End
```



```
Else if INFIX[J]= '(' Then
  Push(s,INFIX[J])
Else if INFIX[J]= ')' Then
  While(s.Data[S.Top]= '(' ) Do
     Pop(s,Element );
     P=p+1;
     POSTFIX[p]=ELEMENT
   End while
If S.Data[S.Top]= '(' Then
  S.Top=S.Top
End
Else
    P=p+1
    Postfix[p]=INFIX[J];
 End
While not Stack Empty(s) Do
    Pop(s.Element);
     P=p+1
   Postfix[p]= Element
```



```
  End while

 End converting

Function PRIRITY (operator : stack data)

// finding the priority of an operator


CASE  OPERATOR OF

'*' , '\' : PRIORITY=2;

'+' , '-' : PRIORITY=1;

'('       : PRIORITY=0;


 End : CASE

End : PRIORITY
```

٢/ الاستدعاء الذاتي Recursive function (Recursion)

على سبيل المثال:

Factorial

N!= 1 if n=0

Nx(n-1)! If n>0

4! = 4*3!

= 4*(3*2!)

= 4*(3*(2*1!))

= 4*(3*(2*(1*0!)))

= 4*(3*(2*(1*1)))

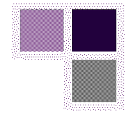
= 4*(3*(2))

= 4*(6)

= 24

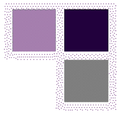
```
Factorial (5) =5*factorial(4)
              =5*(4* Factorial (3))
              =5*(4*(3* Factorial(2)))
              =5*(4*(3*(2* Factorial(1))))
              =5*(4*(3*(2*(1* Factorial(0)))))
              =5*(4*(3*(2*(1*1))))
              =5*(4*(3*(2*1)))
              =5*(4*(3*2))
              =5*(4*6)
              =5*24
              =120
```

```
int factorial (int n)
{
   If (n= = 0) return 1;
Else return n*factorial (n-1);
}
```



There is an almost equally simple iterative version

```
Int factorial (int n)
// factorial iterative version
N is nonnegative integer
Return the value of factorial of n
{
Int count, product =1;
For(count= 1; count<=n; count++)
Product*= count;
Return product;
}
```



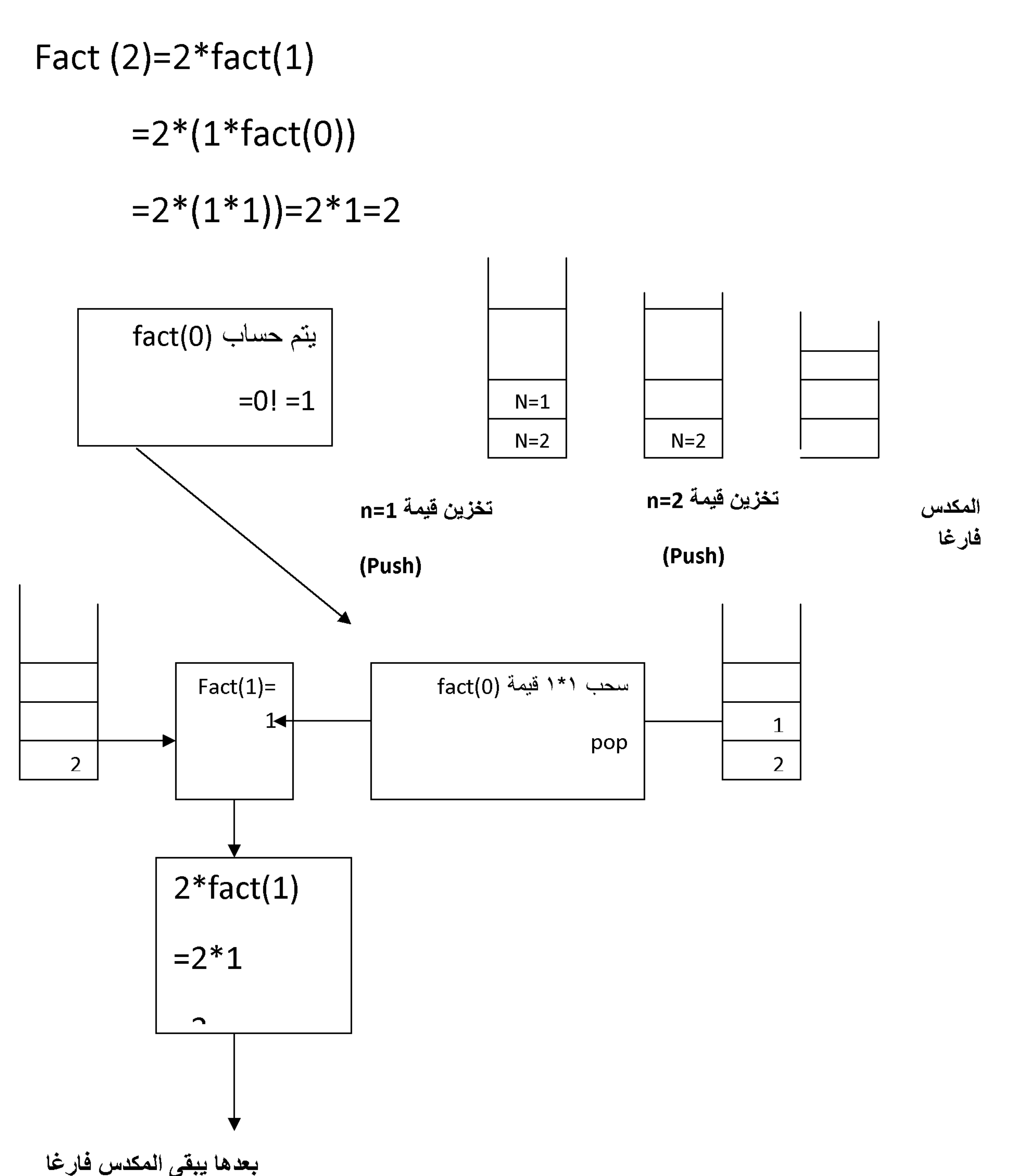
هيئة المكدس عند تنفيذ اقتران الاستدعاء الذاتي

على سبيل المثال

Fact (2)=2*fact(1)

=2*(1*fact(0))

=2*(1*1))=2*1=2

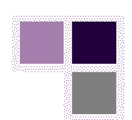


خوارزمية استخدام الـ stack في ايجاد على سبيل المثال fact2

```
Function fact2(n:integer)
F__ integer
S__ stack

Stack creat (s);
Leabel 1: if n=0 then
          F=1;
        Else
          If not stackfull(s) then
            Push (s,n);
                N=n-1;
                Go to label 1;
        While not stack Empty(s) do
                Pop(s,n);
                F=f*n;
             End while;
            Fact2=f
End function;
```

بعدها يبقى المكدس فارغا

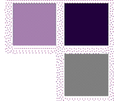


حل مسألة Fibonacci Numbers باستخدام الاستدعاء الذاتي

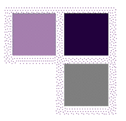
0,1,1,2,3,5,8,13,21,…. ⟶ Fib (n) =fib (n-1) +fib (n-2)

```
Int Fibonacci (int n)
// Fibonacci recursive version
{
If (n<=0) return 0;
Else if (n= =1) return 1;
Else return Fibonacci (n-1)+ Fibonacci(n-2);
}
```

```
Int Fibonacci (int n)
// Fibonacci: iterative version
{int last_but_one // second pervious Fibonacci number fi-2
Int last_value;// Fibonacci number fi-1
Int current; // Fibonacci number fi
If (n <=0) return 0
Else if (n = = 1) return 1;
Else {
last_but_one =0;
```

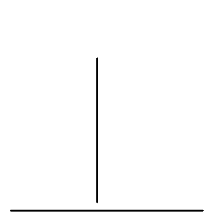


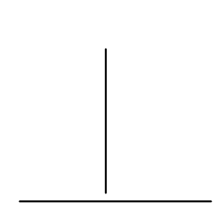
```
last_value=1;
for (int i=2;i<=n;i++){
Current= last_but_one+ last_value;
last_but_one = last_value;
last_value= current;
}
Reyurn current;
}}
```

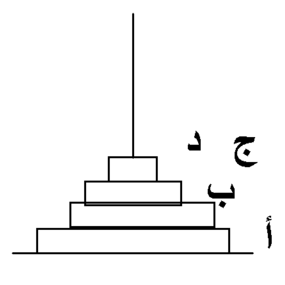


مفهوم الاستدعاء الذاتي مع ابراج هانوي Towers of Hanoi

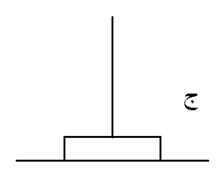
1 2 3



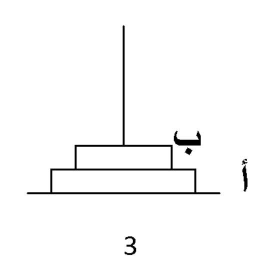




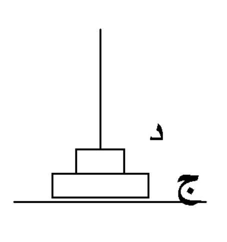
1 2 3

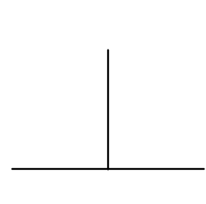


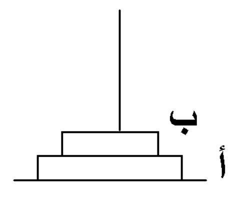
د



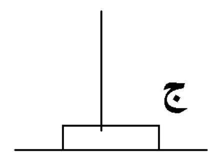
1 2 3

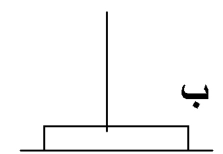


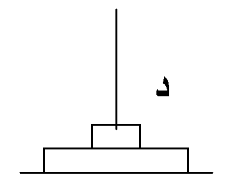


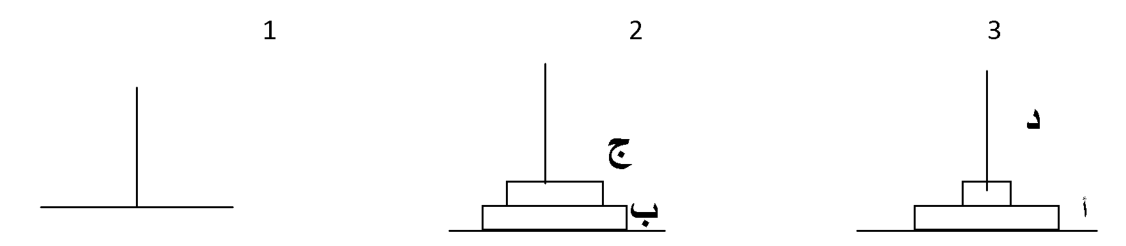


1 2 3









وهكذا حتى يتم تحريك الأقراص من العمود ٣ الى العمود ١ بواسطة استخدام العمود ٢ على شريطة أن تحتفظ هذه الأقراص بالنظام الموجود عليه وان لا يتم نقل أكثر من قرص واحد في المرة الواحده وان لا يوضع أي قرص كبير فوق قرص اصغر منه خلال مرحلة النقل .

برنامج الإستدعاء الذاتي لـ Towers of Hanoi

```
Const int disks=64

// make this constant much smaller to run program

Void move(int count,int start,int finish, int temp),

Main()

{

Move(disks 1,3,2);

// the recursive function that the work is

Void move(int count,int start,int finish, int temp)

{

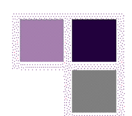
If ( count >0){

Move (count -1,start, temp , finish);

Cout<<" move disk"<< count<<"from"<<start<<"to"<<finish<<"."<<endL;

Move(count-1,temp,finish,start);

}}
```



تمارين

اكتب البرنامج الذي يقوم بإدخال مجموعة قيم إلى داخل المكدس ومن ثم طباعة محتويات المكدس .

```
#include<iostream.h>

Int size=9;

Int a[ 9 ], top= -1;

Int pop();

Void push( int[],int ) ;

Main()

 { int i , k ,;

    For ( i=0 ; i < size ;  ++i )

       { cin >> k ; push( a , k )

    For ( i= 0 ; i< size ; ++i ) cout<<pop() << "  "

     Getch();

}

Void push(  int a[],int k )

{

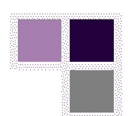
     If ( top==size – 1) cout<< " stack is full " ;

        else

        a[ ++top ] = k;

}

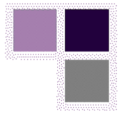
     Int pop()
```



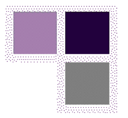
```
{
    If ( top< 0 ) cout<< " stack is empte " ;
       else
     return a[ top - -1];
}
```

اكتب برنامج يقوم بإدخال مجموعه من القيم إلى المكدس ومن ثم القيام بعكس محتويات هذا المكدس .

```
#include<iostream.h>
Int size = 11;
Int pop();
Int a[11],top= -1;
Void w(int[],int )
Void push( int[],int);
Main ()
{
Int i ,k ;
For( i=0; i<size ; i++ ){
Cin>>k; push( a,k) ;
}
W( a )
For ( i= 0 ; i<size ; i++ ){
Cout<< pop()<< " " ;
```



```
Getch() ;
}
void push( int a[] , int k )
{
If ( top==size-1) cout<< " stack is full "
 else
a[ ++top ]= k ;
}
Int pop(){ return a[ top - -1 ] ; }
Void  w( int a[] )
{
Int b[ 11] , c[ 11 ] , top2=-1 , top3=-1;
While ( top>=0) b[++top2]=a[top--] ;
While ( top2>=0 ) c[ ++top3 ]=b[top2--] ;
Whil   ( top3>=0 ) a[++top]=c[top3- -1] ;
}
```



الطابور queues

الطابور عبارة عن قائمة متصلة linked list تقبل الإضافة من مؤخرة الطابور والتي تسمى rear or tail (وهو مؤشر ويقبل الحذف من مقدمة الطابور front or head (وهو مؤشر) .

عملية الإضافة :

max
:
Payroll
Newton
Rear=2
Font=1

Max
:
:
Newton
Rear=1

Max
:
:
Front=1
Rear=0

rear=rear+1 حيث
Queue(rear)=item
Rear=0+1=1
Queue(rear)=item=newton
Rear=rear+1
Queue(rear)=item
Rear=1+1=2
Queue(rear)=item=payroll

عملية الحذف من الطابور :

أنه عندما تتم معالجة العنصر المدخل إلى الطابور فإنه يمكن حذف هذا العنصر من الطابور وهذا يتم بمبدأ irst in first fifo

على سبيل المثال تم معالجة العنصر Newton في الطابور أعلاه فإننا نقوم بحذفه من طرف المقدمة كما يلي :

Item=queue (front)

Front=front1

Newton= queue (front)

Front =1+1=2

max	Payroll	Item → netwton, Rear=2, Front=2

بناء على ما سبق يمكن تحديد الحالات التالية :

الشرط	الحالة
Rear Front	لا يجوز الحذف empty
queue	
Front= Rear	طابور بمدخل واحد
Rear =max	لا يجوز إضافة جديد لان
الطابور ممتلئ	

هناك طريقتين لتمثيل الطابور :

التمثيل التتابعي للطابور أو ما يسمى بالتمثيل باستخدام المصفوفات Queues arraysحيث يتم هنا تمثيل الطابور على شكل مصفوفة أحادية البعد كما يلي :

fron 1	K	R	I	B	y					rear 5

ويقوم هذا المبدأ على تمثيل الطابور بمتتالية بحجم Maxsize ووجود متغيران rearوهو يشير إلى مؤخرة الطابور ولكن قيمته لا تساوي maxsize في هذه الحالة وإنما تتحدد قيمته على ضوء عمليات الإضافة .

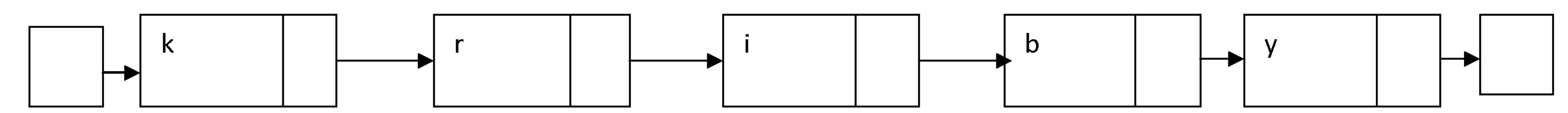
FRoNT:وقيمته تتحدد بمقدار عمليات الحذف .

تمثيل الطابور باستخدام القوائم المتصلة Queues as linked lists

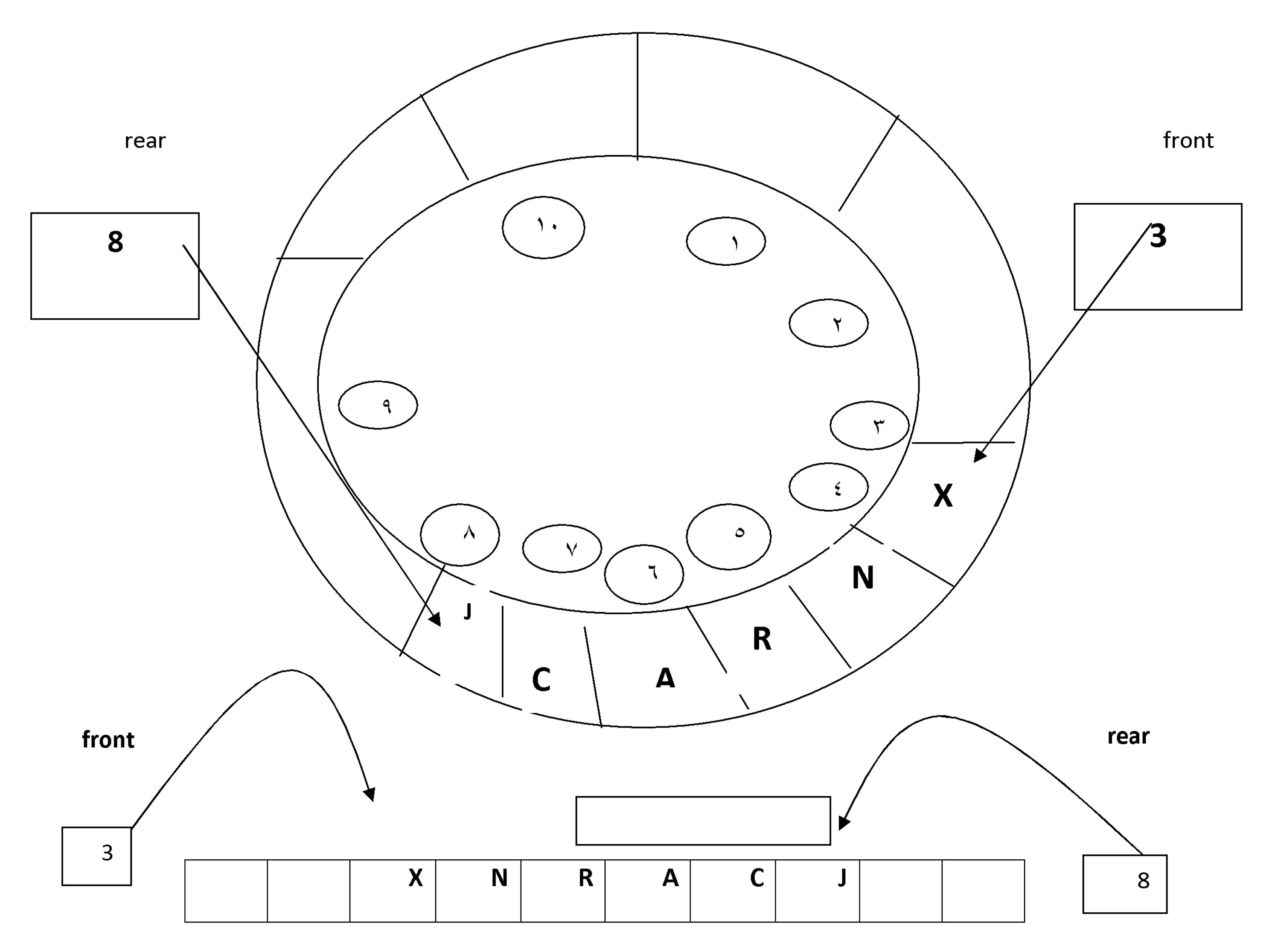
Front

و هنا Rearو Frontتعبران مؤشرين خارجيين .

يختلف هذا النمط عن النمط التتابعي بأنه يحجز في الذاكرة فقط عددا ماديا لعدد القيم التي يشتمل عليها الطابور في لحظة معينة .



ملاحظات : عندما يصل مؤشر الإضافة إلى مقدمة الطابور فإنه لا يمكن إضافة عنصر جديد لطابور الخطي فإنه لا يمكن حذف أي عنصر .
ولذلك ظهرت فكرة إعادة هيكلة ترتيب المصفوفة(الطابور) وذلك بمعاملته كهيكل بياني دائري وهذا باستحداث مؤشرات في المقدمة والمؤخرة تودي بنا إلى تصور ومعاملة المصفوفة على أنها قائمة دائرية circular Queue وبالتالي فإننا نحصل على ما يسمى circular Queue



طوابير الأولوية Priority Queues

من الطبيعي عند معالجة بعض الأعمال بواسطة الحاسوب فإن بعض العمليات يحتاج إلى وقت قليل أما البعض فيحتاج إلى وقت معالجة طويل وبالتالي تظهر مشكلة التفاوت في وقت المعالجة .

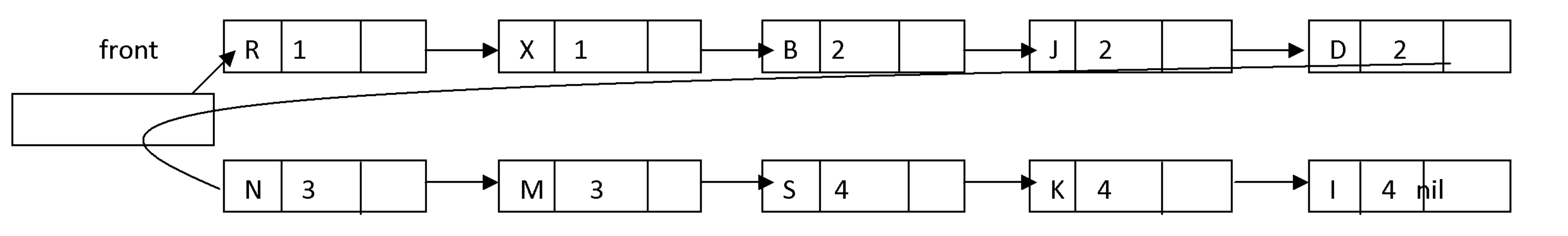
ولذلك ظهر ما يسمى Priority Queues

لنفرض أن لدينا الحالات التالية للتعامل معها :

الشخص	الأولوية	الشخص	الأولوية
B	2	X	**1**
R	١	K	**4**
S	4	M	**3**
J	2	D	**2**
N	**3**	**I**	**4**

لتمثيل هذه الصيغة من الطوابير يمكن استخدام :

الأسلوب الأول : هو تمثيل الطابور على شكل قائمة خطية باستخدام مصفوفة ذات بعد واحد . يمكن استخدام قائمة متصلة مشتركة .

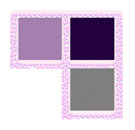


حيث ضمت ثلاث حقول (القيمة) درجة الأولوية ، ومؤشر للعنصر التالي) .

أو يمكن استخدام قائمة مشتركة multi _ linked lists

front ١

1	2	3	4	5	6	7	8	9	10
r	x	b	j	d	n	m	s	K	l
1	1	2	2	2	3	3	4	4	4



الأسلوب الثاني : باستخدام المصفوفات الثنائية : وهنا يخصص لكل فئة من فئات الأولوية صف مستقل وينظر لكل صف على انه يشكل قائمة مستقلة .

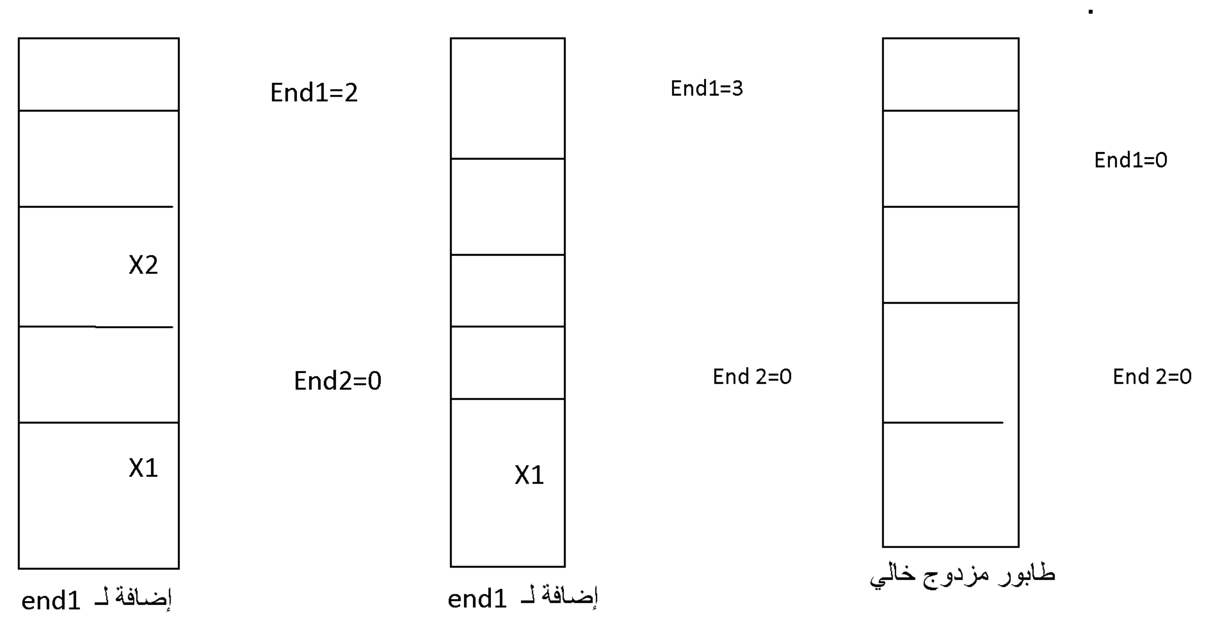
ويتم تخصيص متغيرين لكل صف احدهما للإشارة إلى مؤخرة الطابور لذلك الصف Rear) ويعامل كل صف على انه طابور مستقل) .

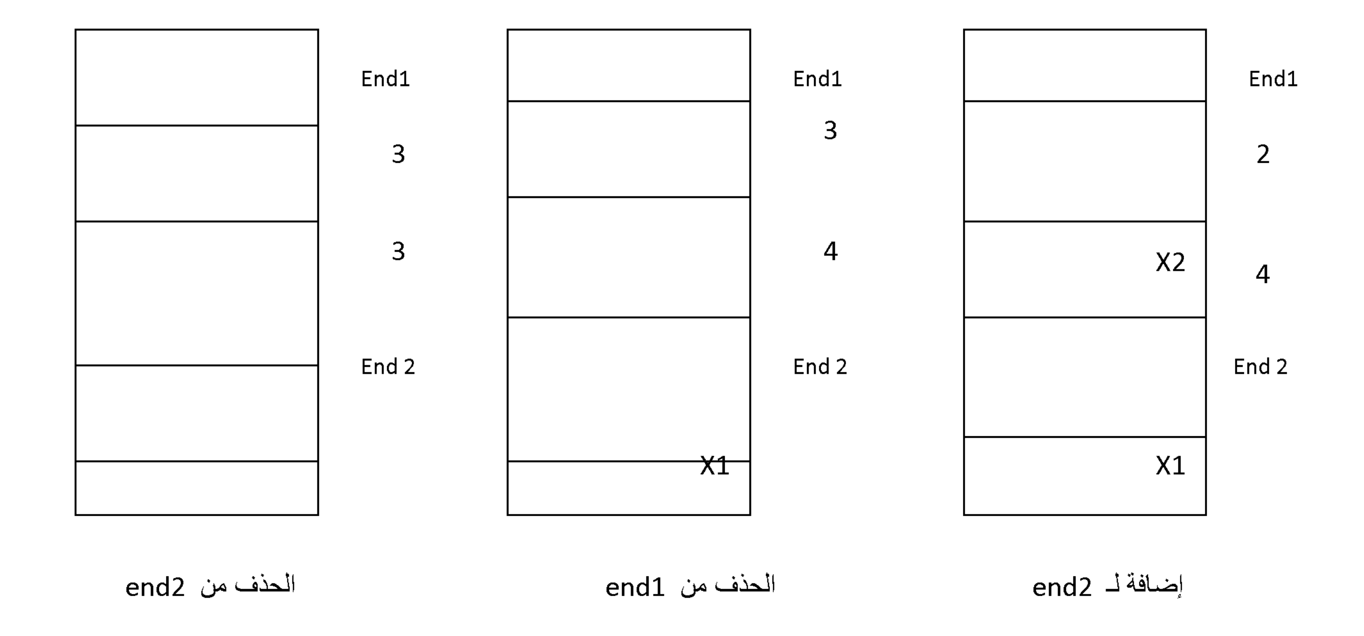
front		1	2	3	4	5	6	7		Rear
1		R	X							2
1		B	J	D						3
1		N	M							2
1		S	K	i						3

الشخص	الأولوية	الشخص	الأولوية
B	٢	X	١
R	١	K	٤
S	٤	M	٣
J	٢	D	٢
n	٣	I	٤

الطوابير المزدوجة double –ended queues(deque)

وهي صيغة عامة أخرى للطوابير إلا أنها تسمح بالإضافة والحذف من كلا الطرفين .





يمكن تمثيل الطوابير المزدوجة باستخدام القوائم المتصلة .

Front

End 1

End2

left

Right



Queue operations in c++

1) Queue::queue( )the queue has been created and initialized to be empty.

2) Error –code queue ::append (const queue- entry

If there is spaes x is added to the queue as its rear .other wise an error –code of over flow is returned.

3)Error_code queue::serve

If the queue is not empty the front of the queue has been removed .otherwise an error code of underflow is returned.

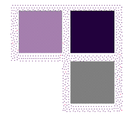
4) Error code queue:: retrieve(queue &x) const if the queue is not empty the front of the queue has been recorded as x .otherwise an Error _code of underflow is returned .

5) bool queue ::empty ()const,

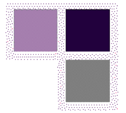
Return true if the queue is empty, otherwise return false.

Linked queue

```
//Basic declarations
Class queue {
Public:
//standard queue methods
Queue ();
Bool empty () const;
Error_code append (const queue entry & item);
Error_code serve ();
Error_code retrieve (queue_entry & item) const;
//safety features for linked structures
~ queue ();
Queue (const queue & original);
Void operator= (const queue & original);
Protected:
Node*front,*rear;
```



```
};
Queue::queue ()
//the queue is initialized to be empty
{
Front = rear =null;
}
//add item to the rear of the queue and return a code of success or return a code of overflow if dynamic memory is exhausted
Error_code queue:: append(const queue_entry & item)
{
Nonde * new_rear==null) return overflow;
If (rear==null) front =rear =new_rear;
Else {
Rear→next=new_rear;
Rear=new_rear;
}
Return success;
}
Error_code queue::serve()
```



//the front of the queue is removed if the queue is empty , return an error_code of underflow

```
{
If(front ==null) return underflow;
Node*old_front =front;
Front=old_front→next;
If(front==null) rear=null;
Delete old_front;
Return success;
}
```

1- consider the following queue of character , where queue is a circular array which is allocated six memory cells:

QUEUE

FRONT=2 , REAR=4

1	2	3	4	5	6
A	C	D			

* describe the queue as the following operations take place:

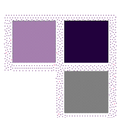
1- f is add to the queue front=2 rear=5

(T.E. f is added to the rear)

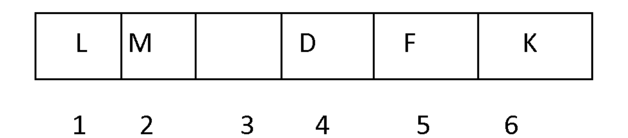
1	2	3	4	5	6
	A	C	D	f	

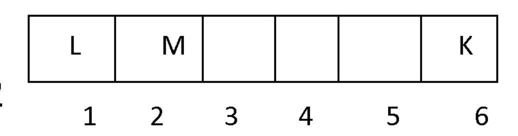
2- A and c are deleted front = 4 rear=5

1	2	3	4	5	6
		D	f		

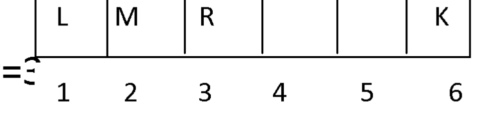


3- K,L and m are added to the rear of queue front=4 rear=2

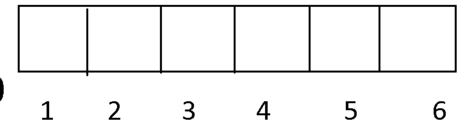




4- D and f are deleted front=6 rear=2



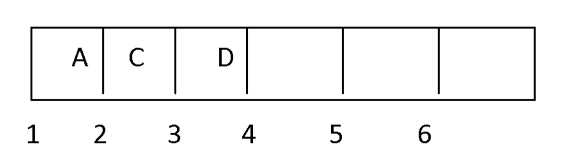
5- R is added to the rear front=6 rear=3



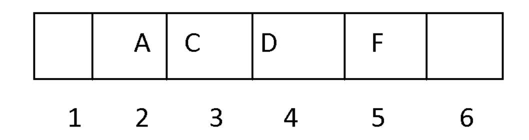
6- All letters are deleted front=0 rear=0

2- Consider the following dequeue of characters where dequeue is circular array which is allocated sex memory cells

Left=2,right=4 dequeue



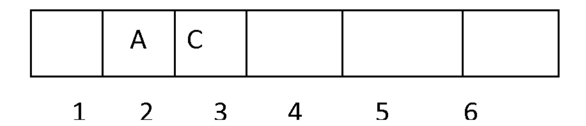
Describe the dequeue while the following operation take place



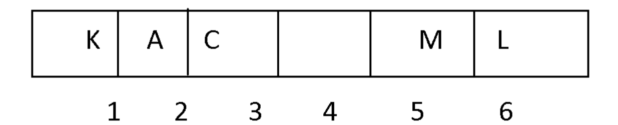
1- f is added on right left=2 right=5

2- f and d are deleted left=2 right=3

Note that the two right letters f and d are deleted so right is decreased by 2



3- K, L, M are added on left left=5 right=3



استخدام القائمة المفردة لتمثيل مصفوفة الأولويات

One-way list representation of a priority matrix

The following figure shows a schematic diagram of a priority queue with 7 elements

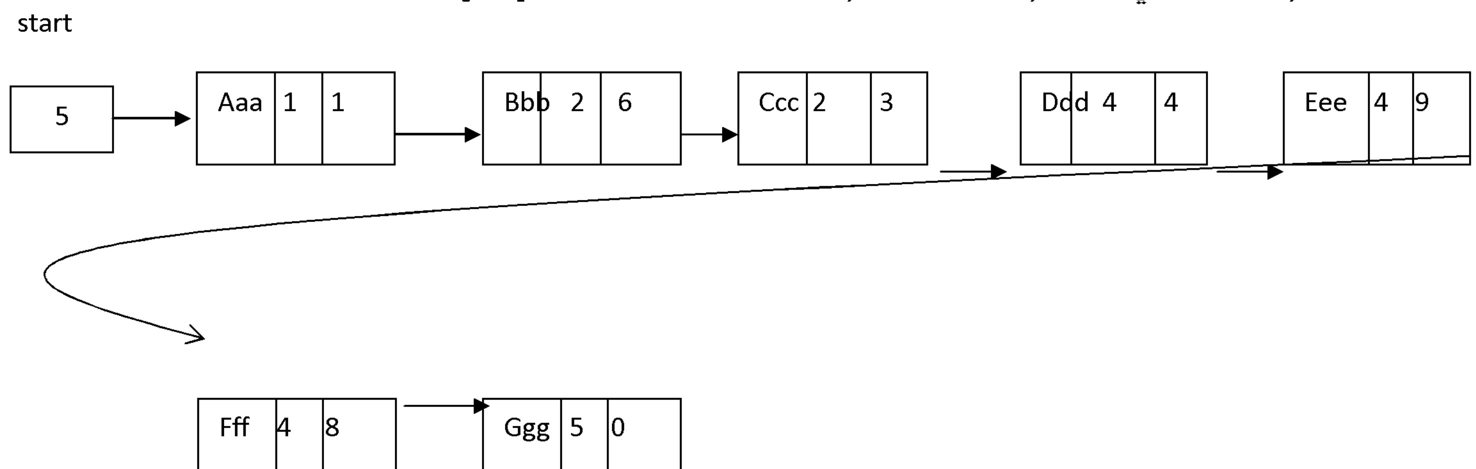
One-way list representation of a priority

الخطوة الأولى إضافة xxx,2 وهي تأتي حسب الأولوية بعد ccc وقبل ddd وبالتالي xxx تخزن في أول خلية فارغة وهي info[2]

الخطوة الثانية إضافة yyy,3 ⟵ inf[7] أي في الخلية الفارغة التالية

الخطوة الثالثة إضافة zzz,2 وتأتي قبل yyy,3 وبعد xxx,2 وتخزن zzz في info[10]

www,1 تأتي قبل bbb,2 وبعد aaa,1 www ⟵ info[11]



start

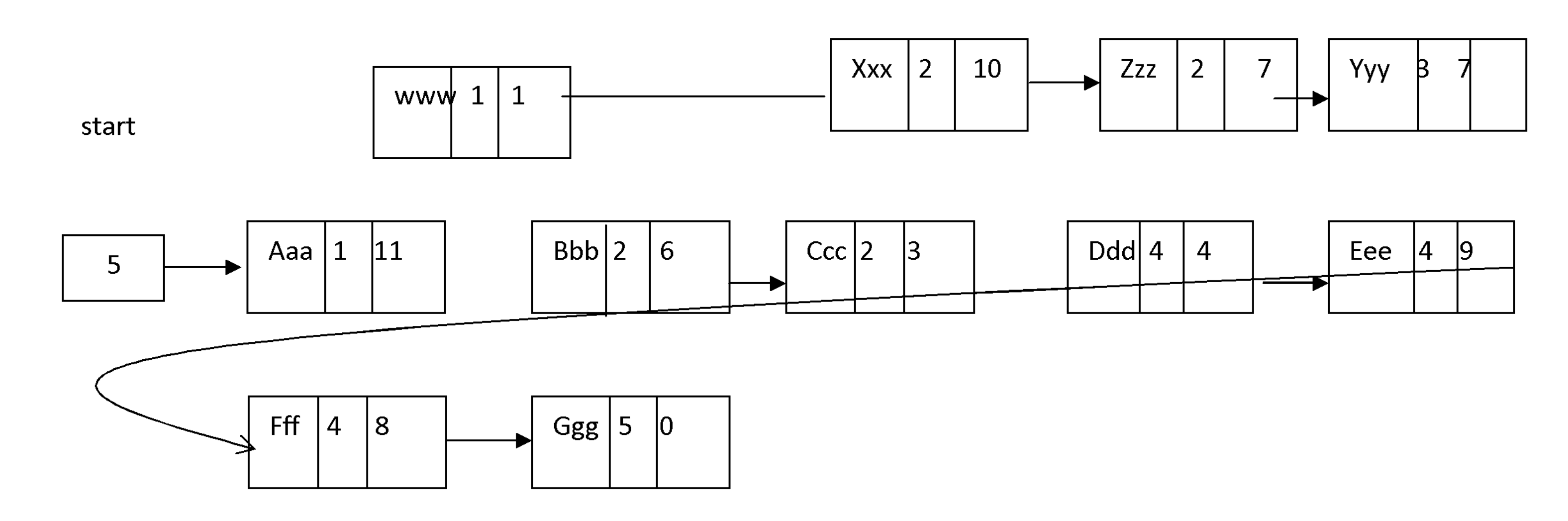
5

free

١2

	Info		Pr		Link
١	Bbb		2		6
٢					
٣	Ddd		4		4
	Eee		4		9
٥	Aaa		1		1
٦	Ccc		2		3
٧					
٨	Ggg		5		0
٩	Fff		4		8
١٠					
١١					
١٢					

Describe the structure after xxx,2 yyy,3 zzz,2 and www,1 are added to the queue



start: 5

free: 12

Bbb		2		6
Xxx		2		10
ddd		4		4
Eee		4		9
Aaa		1		11
Ccc		2		2
Yyy		3		3
Ggg		5		0
Fff		4		8
Zzz		2		7
www		1		1
				0

ب- أحذف aaa,www,bbb من الطابور بعد الإضافة السابقة أعلاه

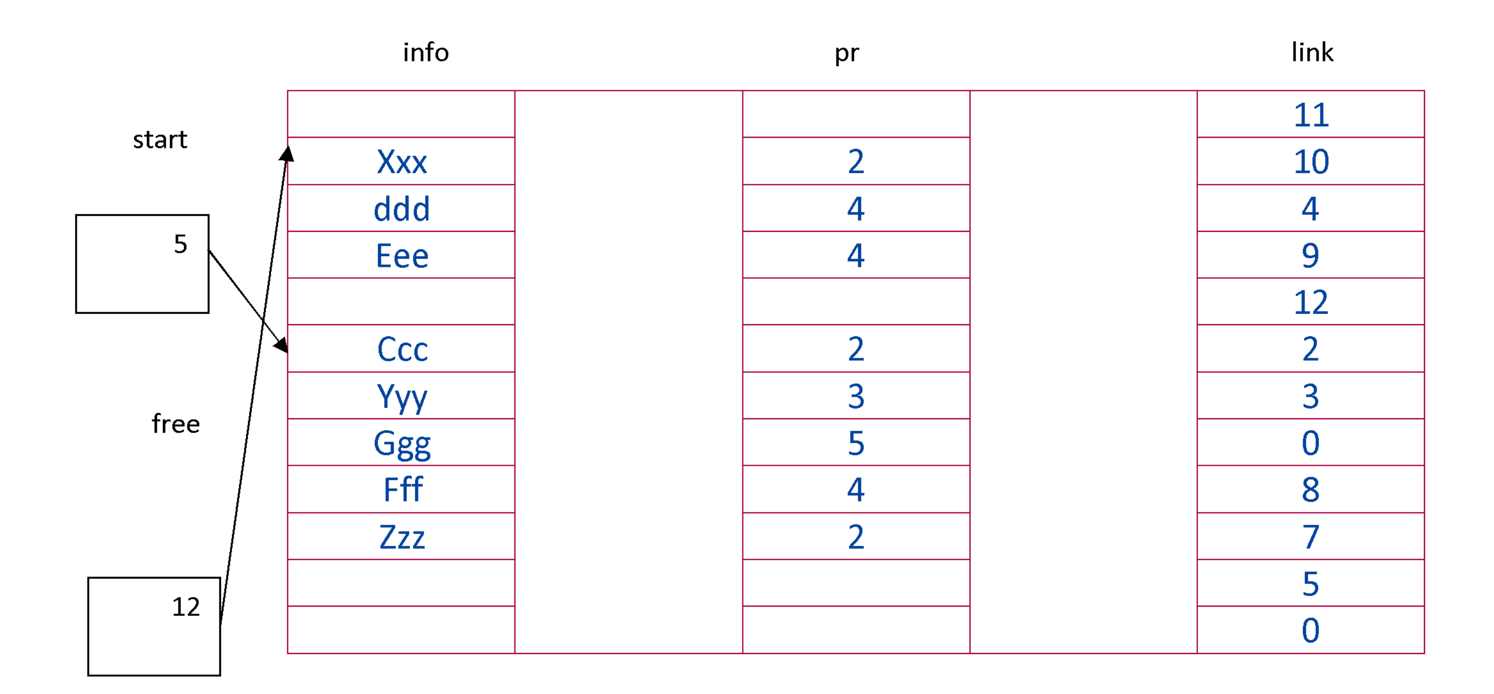
ما يحذف فإن خليته تضاف إلى free

Info[5] ←— aaa

Info[11] ←— www

Info[1] ←— bbb

Describe the structure if, after the preceding insertions, aaa, www, bbb are deleted



info	pr	link
		11
Xxx	2	10
ddd	4	4
Eee	4	9
		12
Ccc	2	2
Yyy	3	3
Ggg	5	0
Fff	4	8
Zzz	2	7
		5
		0

Algorithm deletes and processes the first element in a priority queue which appears in memory as one-way list

1- Set item=info [start][this saves the data in the first node]

2- Delete first node from the list

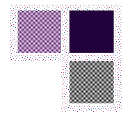
3- Process item

4- Exit

Algorithm adds an item

1- traverse the one-way list until finding a node x whose priority number exceeds n insert item in front of node x

2- If no such node is found insert item as the last element of the list



Array representation of a priority queue

1	2	3	4	5	6
	Aaa				
Bbb	Ccc	Xxx			
Fff				Ddd	Eee
			ggg		

Front		Rear
2		2
1		3
0		0
5		1
4		4

Describe the structure after

Rrr,3 sss,4 ttt,1 uuu,4 vvv,2

1	2	3	4	5	6
	Aaa	Ttt			
Bbb	Ccc	Xxx	Vvv		
Fff	Sss	Uuu		Ddd	Eee
			Ggg		

Front		Rear
2		3
1		4
1		1
5		3
4		4

First delete the element with the highest priority in row1, since row1 contains only two elements aaa and ttt, then the front element in row2 bbb must also be deleted

1	2	3	4	5	6
	Ccc	Xxx	Vvv		
rrr					
Fff	sss	uuu		Ddd	Eee
			ggg		

front		Rear
0		1
2		4
1		1
5		3
4		4

خوارزمية الحذف في array queue

1- [Find the first non empty queue]

Find the smallest k such that front[k]! =null

2- Delete and process the front element in row k of queue

3- Exit

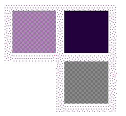
خوارزمية الإضافة في array queue

1- Insert item as the rear element in row m of queue

2- Exit

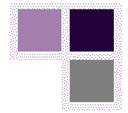
برنامج لتكوين الطابور

```
#include<iostream>
Int size=11 ;
Int a[11] , tail= -1 , head= -1 ;
Int pop_queue() ;
Void() add_queue( int[] , int ) ;
Main()
{
For( i=0 ; i<size ; ++I ){ if( tail==size -1)
{ cout<< " queue is full " ; break ; }
Add_ queue( a , i+1) ;}
While( tail>=head)cout<< pop_queue()<< "    " ;
Getch() ; }
Void add_queue( int a[] , int k )
{  if( tail== -1 ) { head=tail=0 ; a[tail]=k ; }
else
a[++tail]=k ;
}
Int pop_queue() { return a[ head ++ ] ;  }
```

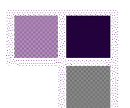


برنامج لعمليات الإضافة والحذف من الطابور

```
#include<iostream.h>
Int size = 11 ;
Int a[11] , tail = -1 , head= -1 ;
Int pop_ queue() ;
Void add_queue( int[] , int ) ;
Void delet( int[] , int ) ;
Main()
{
Int i  ;
For( i=0; size; ++i ) { if( tail==size -1){ cout<< "queue is full";break;}
Add_queue( a , i +1) ; }
Cout<<" enter the value delete\n" ; cin>>i  ;
delet( a , i ) ;
While( tail>=head) cout<<pop_queue()<< "  " ;
Getch() ; }
Void add_queue( int a[] , int k )
{
If( tail== -1 ) { head=tail=0 ; a[tail]=k ;
}
else
a[++tail]=k ;
}
Int pop_queue(){ return a[head++] ; }
Void delet(int a[],int k)
{
int b[11],top2, head2, m=head ; top2=head2= -1 ;
if(tail== -1!! Head>tail)cout<<" queue empte\n"
else
while( m< = tail)  {
 if( a[m] !=k) {
 if(top2== -1) { head2=top2=0; b[top2]=a[m] ; }
   else
b[++top2]=a[m] ;
}
M++;
}
```



```
Head=tail= -1 ;
While( top2>= head2 )
{
If( tail== -1)
{
Head=tail-0 ; a[tail]= b[head2 ] ;
}
else
a [ ++tail = b[head2] ;
head2++ ;
}
}
```



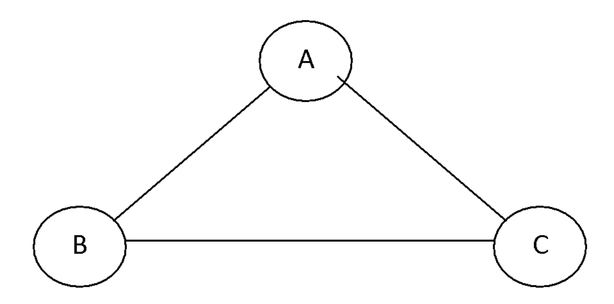
المخططات (Graphs)

ان أي مخطط عادةَ يتكون من جزئيين رئيسين هما:-

١- مجموعة من العناصر (elements or nodes) و يطلق عليهل باسم رؤوس (vertices) ويمز لها V .

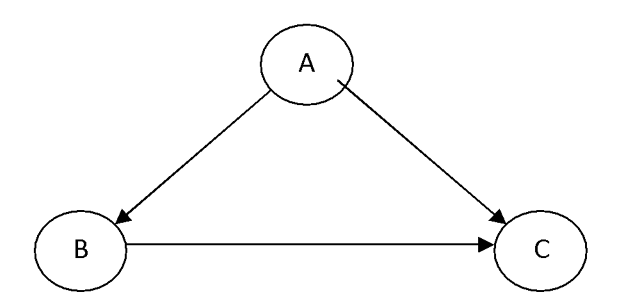
٢- مجموعة من الحواف (Edges) وكل حافة تربط بين عنصرين والمخططات تتواجد على عدة أنواع وهي:-

١-المخطط الغير متجه (undirected graph) مثل الشكل التالي/

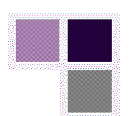


بمعنى أنه على سبيل المثال بأن الحافة (A,B) تكافئ (B,A) وبالتالي فلا داعي لتكرار الحافتين ويمكن ذكر واحدة منها.

٢-المخطط المتجه(Directed Graph) مثل الشكل التالي/



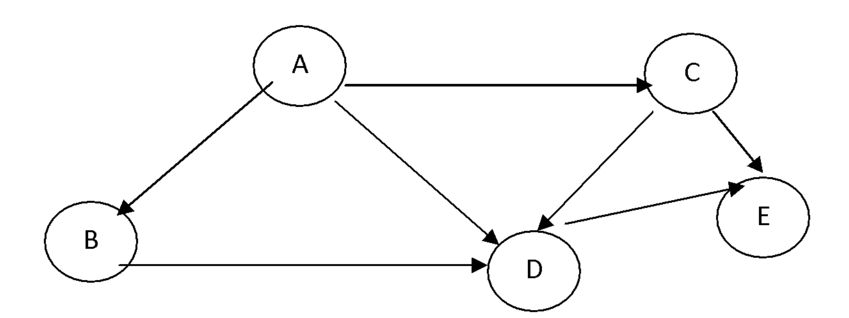
بمعنى أنه يوجد اتجاهات للحواف يشار لها بعلاقة الاتجاه وبالتالي فإن الحافتين (A,B)لا تكافئ (B,A) وعليه وحسب الشكل فإن الحافة (A,B) تمثل حافة من حواف المخطط لوجود إتجاه للحافة ولكن الحافة(B,A) لا تمثل حافة في المخطط لعدم وجود اتجاه لهذه الحافة.



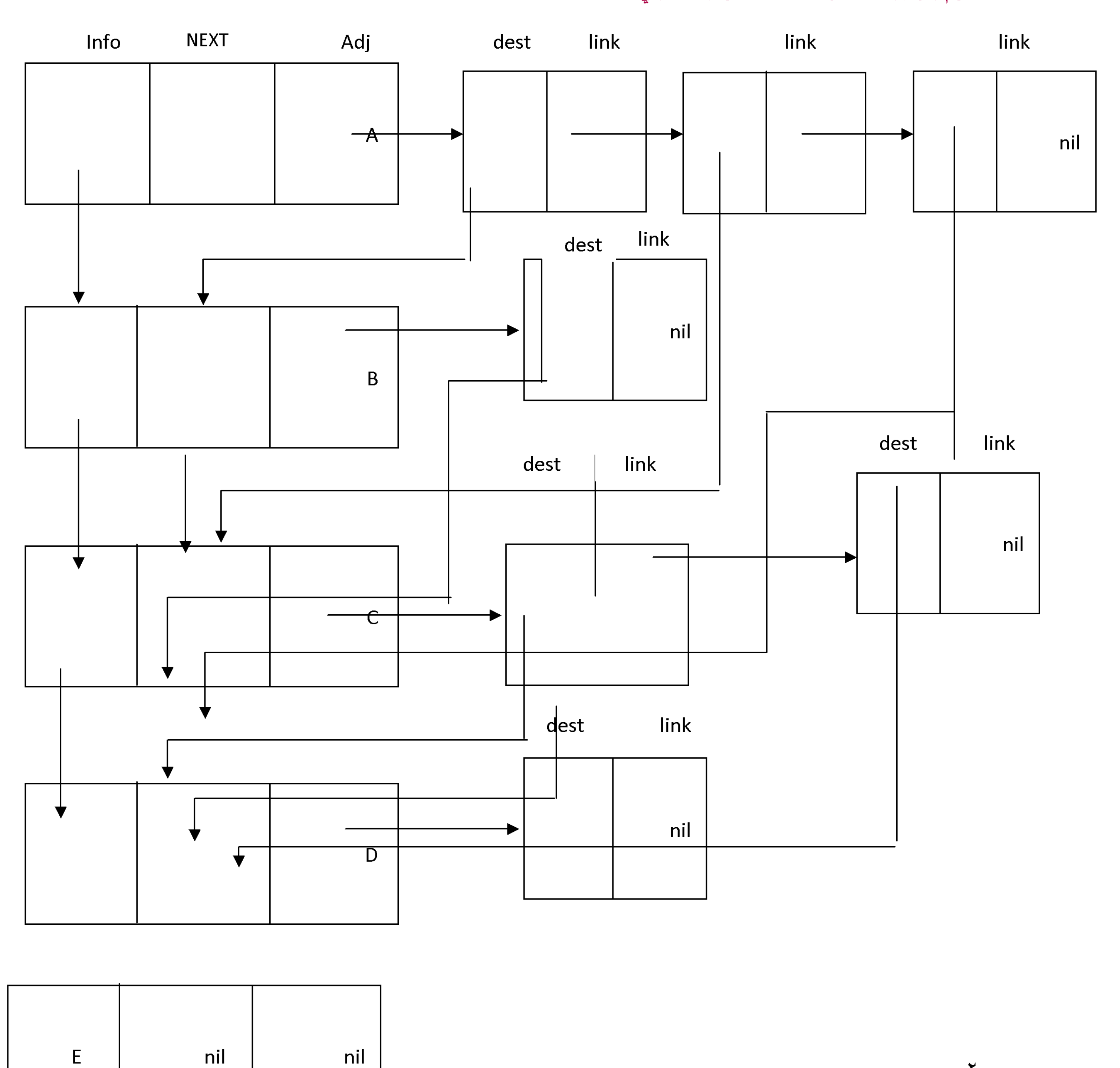
لتمثيل المخططات يوجد طريقتين رئيسيتين:

١-تمثيل المخطط بواسطة مؤشرات الربط (Linked representations).

افرض لدينا المخطط التالي:-



نقوم بترتيب عناصر المخطط عموديا كما يلي:

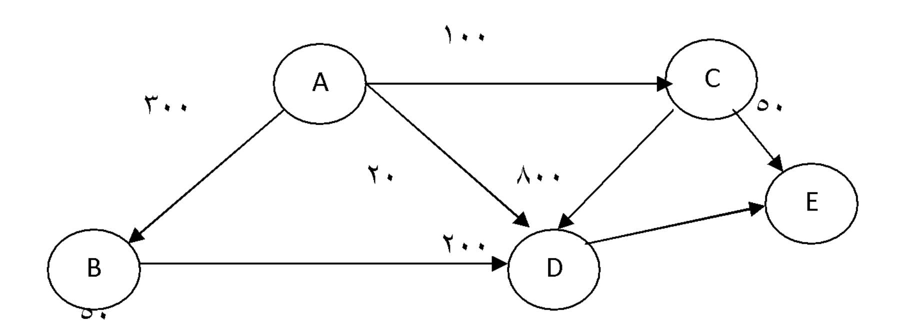


٢

-تمثيل المخطط بواسطة مصفوفة الجوار(Adjacency Matrices) للمخطط السابق نستنتج مصفوفة الجوار:

	A	B	C	D	E
A	0	1	1	1	0
B	0	0	0	1	0
C	0	0	0	1	1
D	0	0	0	0	1
E	0	0	0	0	0

عند إضافة أوزان أو تكلفه إلى حواف المخطط فان المخطط عندها يسمى بالشبكة(NEYWORK) وهذه الأوزان تسمى أوزان الحواف على سبيل المثال:



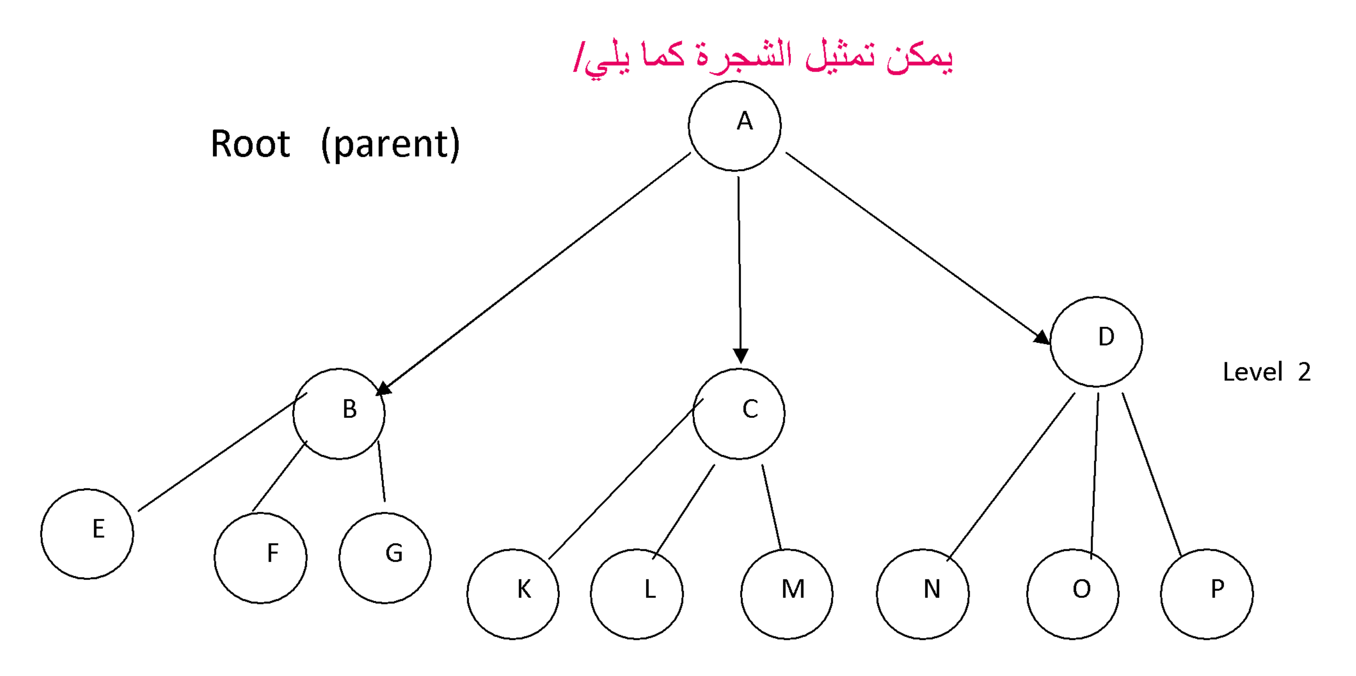
عندها يمكن كتابة ما يسمى بمصفوفة أوزان الجوار (Cost Adjacency) كما يلي...

	A	B	C	D	E
A	0	300	100	20	0
B	0	0	0	50	0
C				800	50
D	0	0	0	0	200
E	0	0	0	0	0

الأشجار (Trees)

الشجرة/هي عبارة عن تركيب هرمي للبيانات وهي تعبر عن حالة خاصة من المخططات.

يمكن تمثيل الشجرة كما يلي/



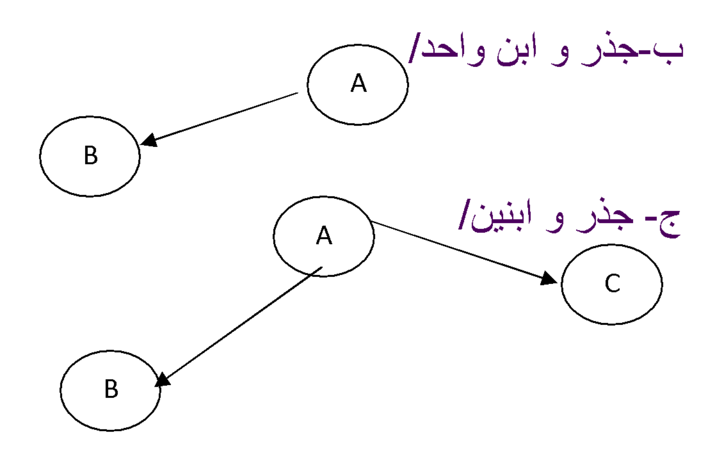
إن محتوى المستوى الأول يعتبر parent المحتويات المستوى الثاني بينما محتوى المستوى الثاني فيعتبر children لمحتويات المستوى الأول وكذلك parent لمحتوى المستوى الثالث

أنواع الأشجار

١-الأشجار الثنائية Binary Tree

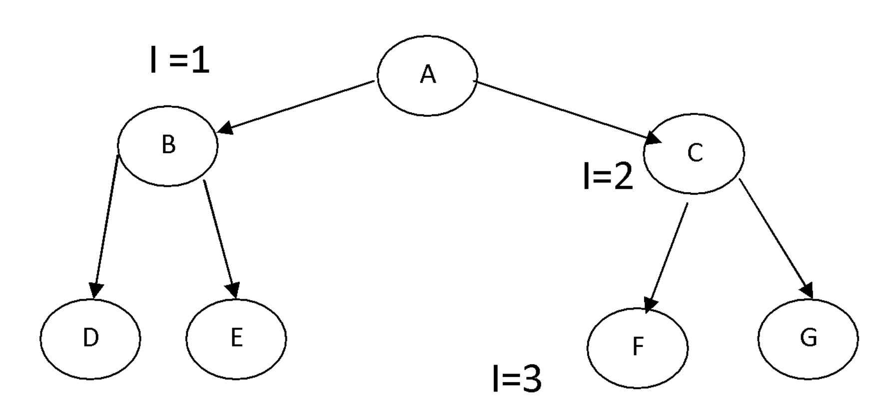
أ-جذر دون أبناء/ A

ب-جذر و ابن واحد/



ج- جذر و ابنين/

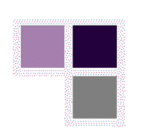
د- الشجرة الثنائية /



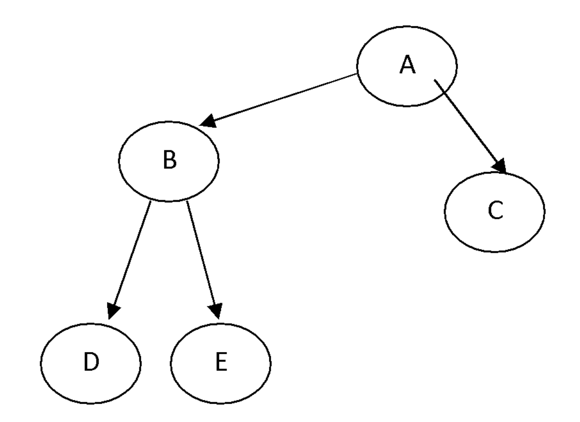
$$2^{i} - 1 \quad = 2^{3} - 1 = 7$$

وهي تسمى شجرة ثنائية متوازنة وتنطبق عليها المعادلة /

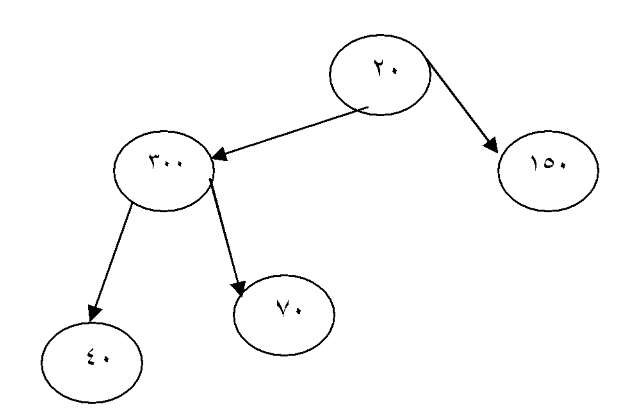
عدد عناصر الشجرة $= 2^{i} - 1$



هـ - شجرة ثنائيه غير كاملة (uncompleted binary tree)/



٢-أشجار القيمة (Value tree) أوالأشجار المقيدة (head tree) لأنه عادة يستخدم فيها الفرز المقيد/



٣-أشجار العمليات(operations tree)/

+
-
*
D
E
E

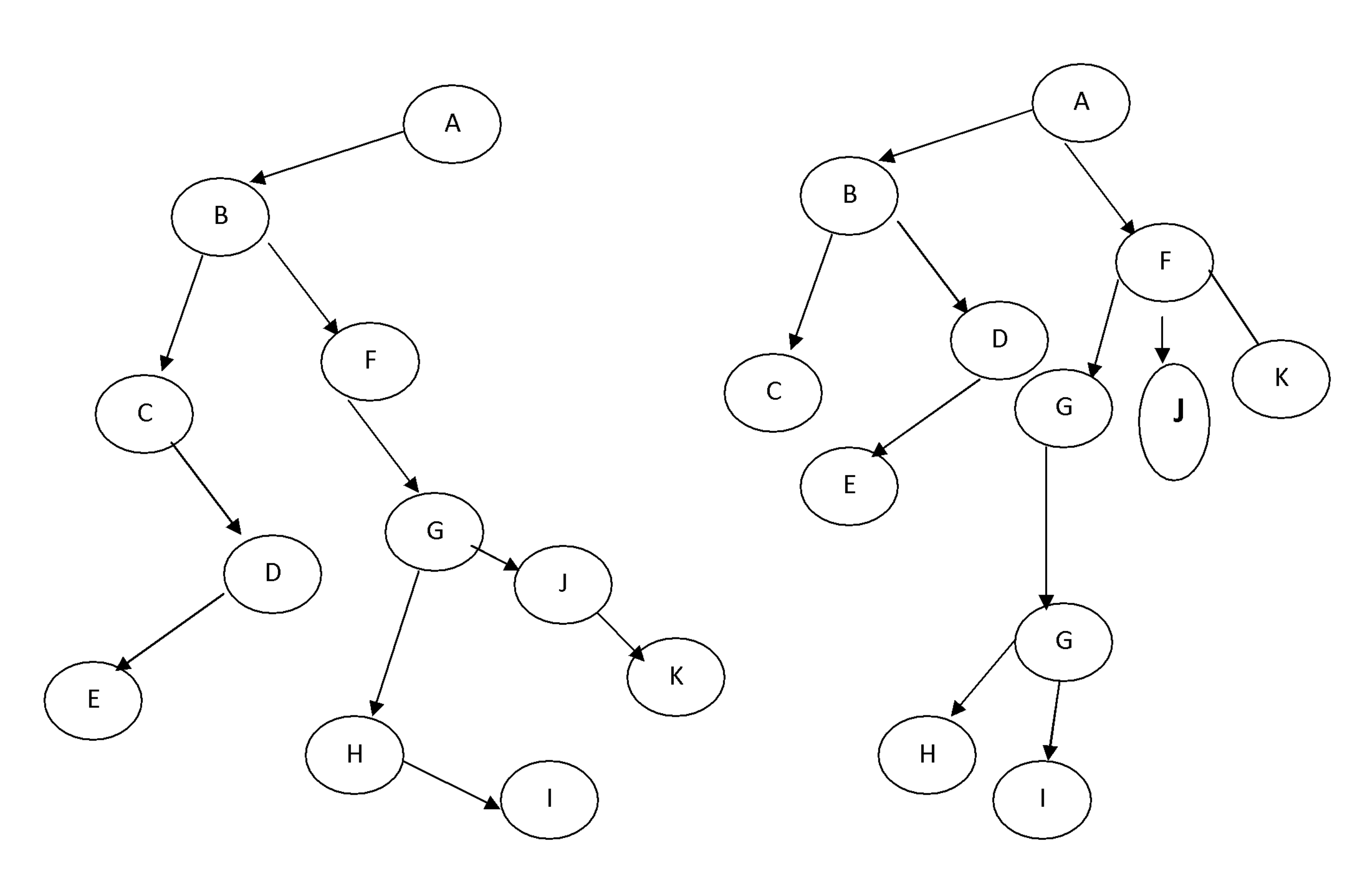
التحويل من شجرة اعتيادية الى شجره ثنائية

١- جذر الشجرة الثنائية هو نفس جذر الشجرة الاعتيادية .

٢- يمكن الحصول على أطفال الشجرة كما يلي/

أ- نأخذ الطفل الأول من جهة اليسار في الشجرة الاعتيادية ونجعله الطفل الأيسر للجذر الموجود(إذا كان موجودا).

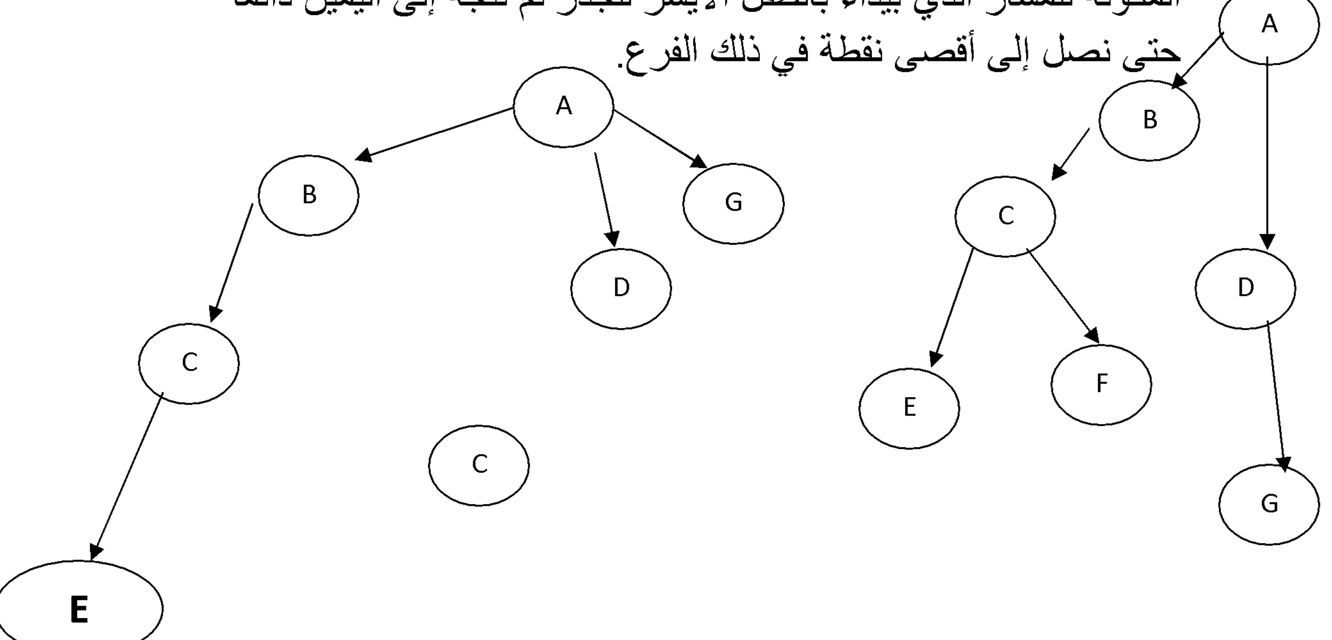
ب-إذا كان هناك نقاط في الشجرة المعطاة لها نفس مستوى النقطة المأخوذة كطفل أيسر فإننا نعتبر النقطة الأولى القريبة منها كطفل ايمن للنقطة التي أخذت كطفل أيسر في خطوة (أ).



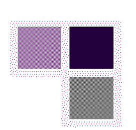
شجرة ثنائية شجره اعتيادية

التحويل من شجرة ثنائية الى شجرة اعتيادية

١- جذر الشجرة الاعتيادية هو نفس جذر الشجرة الثنائية.
٢- الأطفال المباشرين للجذر في الشجرة الاعتيادية يمكن الحصول عليهم من الفرع الأيسر للشجرة الثنائية المعطاة،حيث يكون الأطفال عبارة عن النقاط المكونة للمسار الذي يبداء بالطفل الأيسر للجذر ثم نتجه إلى اليمين دائما حتى نصل إلى أقصى نقطة في ذلك الفرع.



شجرة اعتيادية شجرة ثنائية



تمثيل المصفوفة باستخدام مصفوفة ذات بعد واحد

(Linear Representation)

لنفرض ان لدينا المعادلة الحسابية

(A-B)+C*(D/E)

يمكن تمثيل هذه المعادلة على شكل شجرة ثنائية كما يلي

[i=1]

level i=1

+

-

[2*i=2]

*

[2*i+1=3]

i=2

B

A

2*3+1=7]

[2*i=2*2=4]

C

[2*i+1=5]

/

i=3

D

[2*3=6]

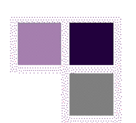
E

[2*7=14]

[2*7+1=15] i=4

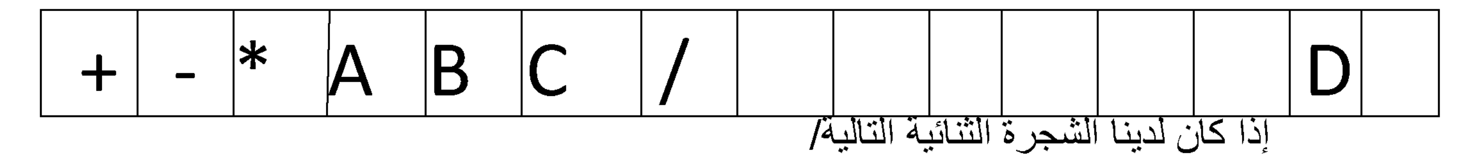
وحتى تكون هذه الشجرة متوازنة وكاملة فيجب ان تكون عدد عناصرها $2^i-1=$

وبالتالي $4^2 - 1=15$

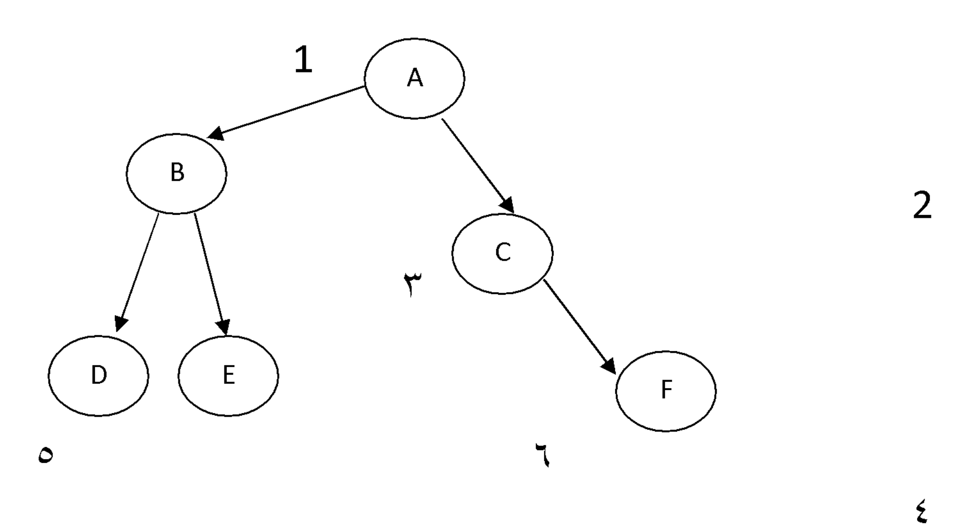


أي انها تحتوي على ١٥ عنصر وعليه فان المصفوفة احادية البعد ستمثل كما يلي/

1 2 3 4 5 6 7 8 9 10
11 12 13 14 15



إذا كان لدينا الشجرة الثنائية التالية/



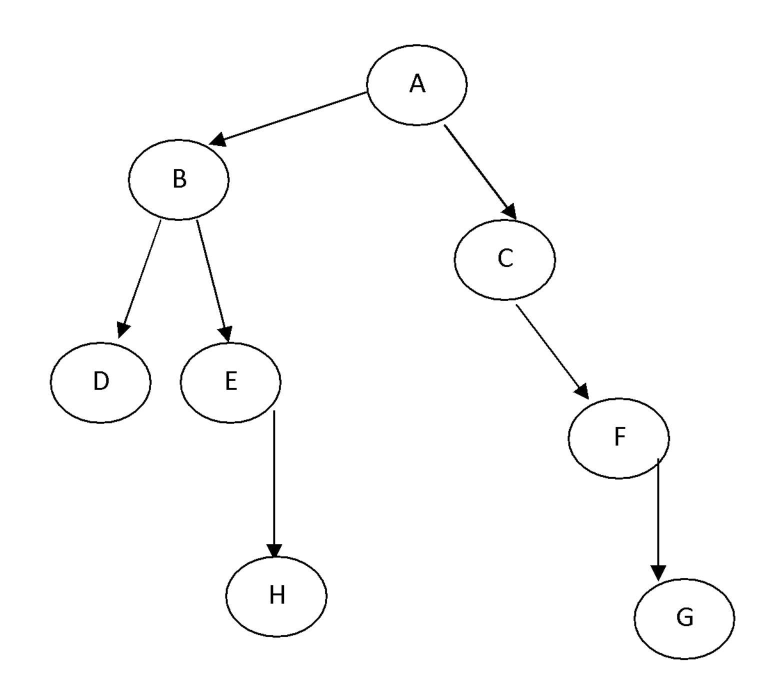
فيمكن تمثيلها بمصفوفة الحواف كما يلي/

Index	Info	Left	Right
1	A	2	3
2	B	4	5
3	C	6	0
4	D	0	0
5	E	0	0
6	F	0	0

استعراض الأشجار الثنائية(Binary Tree Traversal)

يوجد ثلاث أنواع من الطرق استعراض الأشجار الثنائية.

لنفرض أن لدينا الشجرة الثنائية التالية/



1-Preorder traverse:

a) Visit the root
b) Visit the lift sub tree
c) Visit the right sub tree

Solution:- ABDEHCFG

2-Postorder traverse:-

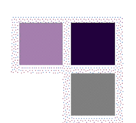
a) Visit the lift sub tree
b) Visit the right sub tree
c) Visit the root

Solution: - DHEBFGCA

3-Inorder traverse:

a) Visit the lift sub tree
b) Visit the root
c) Visit the right sub tree

Solution: - DHEBFGCA

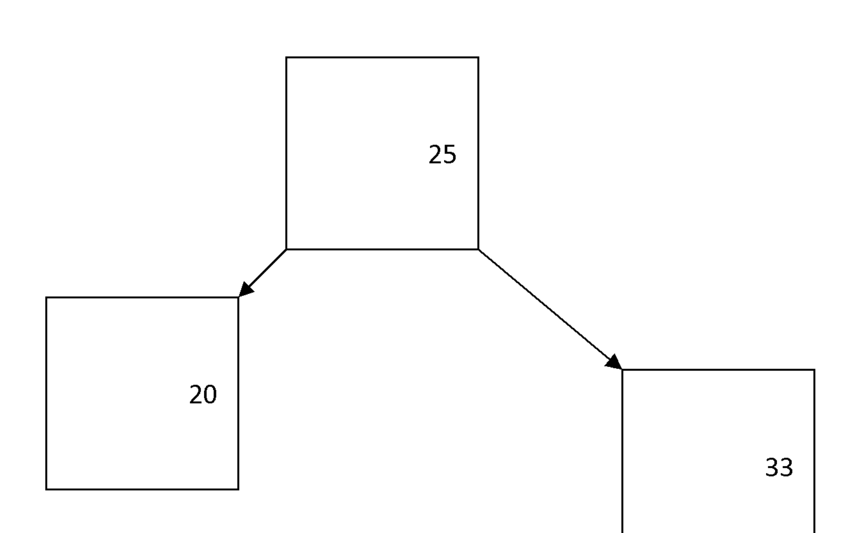


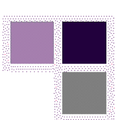
برنامج بناء الشجرة الثنائية:

افرض انه معطى المتتالية (٣٣ ، ٢٠ ، ٢٥) لبناء الشجرة الثنائية :-

١. اعتبر العنصر الأول هو الجذر

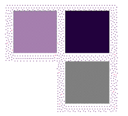
٢. بالنسبة للعدد التالي فإذا كان اكبر من الجذر نضعه من يمين الجذر والا نضعه من يساره





ويلي برنامج بناء الشجرة الثنائية

```
#include<iostream.h>
Struct  tree { int d , tree * left , tree * right ; }
Void insert ( tree * , tree * ) ;
Print( tree * ) ;
Void main()
{
tree *node , *s , *p , *root=nill ;
for( int i=1 ; i< 2 ; ++I ) {
  node=new tree ;
cin>>nod→d ; node→left=node→right=0 ;
if ( root== nill ) root= node ;
else
insert( root , node ) ; }
print( root )
getch() ; }
void insert ( tree *root , tree *node) {
tree *p , *s ;
s=root ;
while( s != nill )
{
p=s; if(node→d > s→d) s=s→right ; else s=s→left ;
}
If( node→d > p→d) p→right=node ; else p→left=node ;
}
Print ( tree *node)
{
If( node !=nill){
Print( node→left) ;
Print(node→right) ;
Cout<< node→d << "  "
}}
```



تمارين وحلول

III. STACKS

Stacks are commonly used Data Structures while writing code. It's concept is
really simple which makes it even simpler to write it in code. Consider this
situation. There are a pile of 5 Books on a Table. You want to add one book to
the pile. What do you do??? You simply add the book on the TOP of the pile. What
if you want the third book from the new 6 book pile? You then lift each book one
by one from the TOP until the third book reaches the top. Then you take the
third book and replace all the others back into the pile by adding them from the
TOP.

If you've noticed I've mentioned the word TOP in Caps. Yes, TOP is the most
important word as far as stacks are concerned. Data is stored in a Stack where
adding of data is permitted only from the top. Removing/Deleting Data is also

done from the top. As Simple as That. Now you may ask where Stacks are used?

Stacks are infact used on every Processor. Each processor has a stack where data

and addresses are pushed or added to the stack. Again the TOP rule is followed

here. The ESP Register adds as a Stack Pointer that refers to the top of the

stack in the Processor. Anyway, since the explaination of how the Processor

Stack works is beyond the subject of this Tutorial, let's write our Stack Data

Structure. Remember some Stack Terminology before continuing. Adding Data to the

Stack is known as Pushing and deleting data from the stack is known as Popping.

cpp

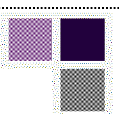
```
#include <iostream>

 using namespace std;

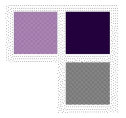
 #define MAX 10        // MAXIMUM STACK CONTENT

 class stack

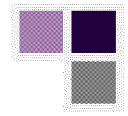
{
```



```
private:
 int arr [MAX];   // Contains all the Data
int top;       //Contains location of Topmost Data pushed onto Stack
 public:
   stack ()        //Constructor
{
 top=-1;      //Sets the Top Location to -1 indicating an empty stack
   }
void push(int a)  // Push ie. Add Value Function
{
    top++;       // increment to by 1
  if(top<MAX)
   {
arr[top]=a;  //If Stack is Vacant store Value in Array
     }
 else          {               cout<<"STACK FULL!!"<<endl;
        top--;
      }
```



```
}
 int pop()                // Delete Item. Returns the deleted item
  {
  if(top==-1)
  {
cout<<"STACK IS EMPTY!!!"<<endl;
         return NULL;
 }
  else
  {
  int data=arr[top];     //Set Topmost Value in data
        arr[top]=NULL;       //Set Original Location to NULL
   top--;             // Decrement top by 1
      return data;         // Return deleted item
  }
   }
};
int main()
```



```
{
stack a;
a.push(3);
cout<<"3 is Pushed\n";
 a.push(10);
 cout<<"10 is Pushed\n";
 a.push(1);
 cout<<"1 is Pushed\n\n";
cout<<a.pop()<<" is Popped\n";
 cout<<a.pop()<<" is Popped\n";
 cout<<a.pop()<<" is Popped\n";
 return 0;
}
```

OUTPUT:

3 is Pushed

10 is Pushed

1 is Pushed

1 is Popped

10 is Popped

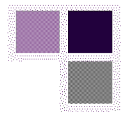
3 is Popped

Clearly we can see that the last data pushed is the first one to be popped out.
That's why a Stack is also known as a LIFO Data Structure which stands for "Last
In,First Out" and I guess you know why.
Let us see how we implemented the stack. We first created a variable called top
that points to the top of the stack. It is initialised to -1 to indicate that
the stack is empty. As Data is entered, the value in top increments itself and
data is stored into an array arr. Now there's one drawback to this Data
Structure. Here we state the Maximum number of elements as 10. What if we need

more than 10 Data Elements? In that case we combine a Stack along with a Linked
List which will be explained later.
Now once you've got this one right, let's proceed to the Queue Data Structure.

IV. QUEUES

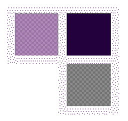
There's a huge crowd at your local grocery store. There are too many people
trying to buy their respective items and the Shopkeeper doesnt know from where
to start. Everyone wants their job done quickly and the shopkeeper needs an
efficient method to solve this problem. What does he do? He introduces a Queue
System based on the First Come, First Serve System. The Last Person trying to
buy an item stands behind the last person at the END of the queue. The
Shopkeeper however is present at the FRONT end of the queue. He gives the item
to the person in FRONT of the queue and after the transaction is done, the
person in FRONT of the Queue Leaves. Then the person



second in queue becomes the
First person in the Queue.

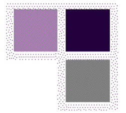
Do you get the point here? A Queue is similar to a stack except that addition of
data is done in the BACK end and deletion of data is done in the FRONT.
Writing a queue is a lot harder than writing a stack. We maintain 2 Integers in
our Queue Data Structure, one signifying the FRONT end of the Queue and the
other referring to the BACK end of the Queue.

Let us use the same coding style as we used for the Stack. We first initialise
both the ends to -1 to indicate an empty queue. When Data is added to the queue
both ends get respective postive values. When New Data is added, the back End is
incremented and when data is deleted the front end is decremented. This works
fine but it has a major drawback. What if the Maximum capacity of the Queue is 5
Items. Suppose the User has added 4 items, deleted 3



items and adds 2 again. The
Stack wont permit him to add the last half of the data as it will report that
the stack is full. The Reason is that we are blindly incrementing/decrementing
each end depending on addition/deletion not realising that both the ends are
related to each other. I leave this as an excercise for you to answer. Why will
our proposed Stack report the Stack as Full even though it's actually vacant?
So we need another approach.In this method we focus more on the data than on the
addition end and the deletion end.

What we now use is the grocery store example again. Suppose there are 5 items in
a queue and we want to delete them one by one. We first delete the first data
item pointed by the deletion end. Then we shift all data one step ahead so that
the second item becomes the first, third item becomes second and so on...
Another method would be to maintain the difference



between the two ends which is
not practical. Hence we stick to our previous method. It might be slow in Big
Queues, but it certainly works great. Here's the code.

cpp

```
#include <iostream>

 using namespace std;

 #define MAX 5           // MAXIMUM CONTENTS IN QUEUE

 class queue

 {

 private:

   int t[MAX];

    int al;      // Addition End

  int dl;      // Deletion End

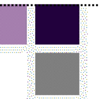
public:

  queue()

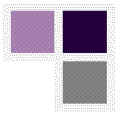
  {

  dl=-1;

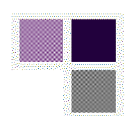
    al=-1;
```



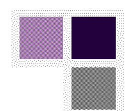
```
 }
void del()
 {
 int tmp;
   if(dl==-1)
 {
      cout<<"Queue is Empty";
   }
   Else
   {
      for(int j=0;j<=al;j++)
      {
         if((j+1)<=al)
        {
          tmp=t[j+1];
             t[j]=tmp;
         }
         else
```



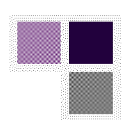
```
          {
            al--;
          if(al==-1)
            dl=-1;
           else
             dl=0;
          }
        }
     }
  }
void add(int item)
{
   if(dl==-1 && al==-1)
{
      dl++;
      al++;
   }
  else
```



```
{
    al++;
     if(al==MAX)
{
           cout<<"Queue is Full\n";
            al--;
            return;
        }
   }
t[al]=item;
}
  void display()
  {
    if(dl!=-1)
  {
   for(int iter=0 ; iter<=al ; iter++)
     cout<<t[iter]<<" ";
   }
```



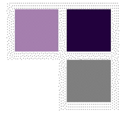
```
 else     cout<<"EMPTY";
 }
};
int main()
{
queue a;
int data[5]={32,23,45,99,24};
cout<<"Queue before adding Elements: ";
a.display();
 cout<<endl<<endl;
for(int iter = 0 ; iter < 5 ; iter++)
{
a.add(data[iter]);
  cout<<"Addition Number : "<<(iter+1)<<" : ";
  a.display();
   cout<<endl;
}
cout<<endl;  cout<<"Queue after adding Elements: ";
```



```
a.display();
cout<<endl<<endl;
for(iter=0 ; iter < 5 ; iter++)
 {
   a.del();
  cout<<"Deletion Number : "<<(iter+1)<<" : ";
   a.display();
   cout<<endl;
 }
 return 0;
}
```

OUTPUT:

Queue before adding Elements: EMPTY



Addition Number : 1 : 32

Addition Number : 2 : 32 23

Addition Number : 3 : 32 23 45

Addition Number : 4 : 32 23 45 99

Addition Number : 5 : 32 23 45 99 24

Queue after adding Elements: 32 23 45 99 24

Deletion Number : 1 : 23 45 99 24

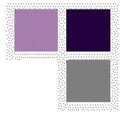
Deletion Number : 2 : 45 99 24

Deletion Number : 3 : 99 24

Deletion Number : 4 : 24

Deletion Number : 5 : EMPTY

As you can clearly see through the output of this program that addition is
always done at the end of the queue while deletion is done from the front end of
the queue. Once again the Maximum limit of Data will be



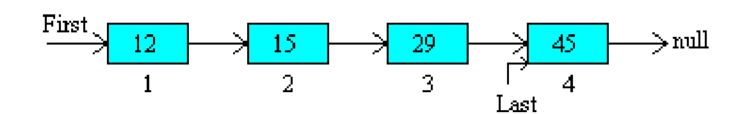
extended later when we
learn Linked Lists.

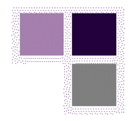
V. LINKED LISTS

The Linked List is a more complex data structure than the stack and queue. A
Linked List consists of two parts, one the DATA half and the POINTER half. The
Data half contains the data that we want to store while the pointer half
contains a pointer that points to the next linked list data structure. This way
we have a dynamic data structure as we can add as much data as we want within
memory restrictions. And yes, pointers play a major role in Data Structures...No
Pointers, No Data Structures...So Knowledge of Pointers is a basic must before
continuing.

Look at this diagram that explains the Linked List:

Items Added in Link List : 12 15 29 45





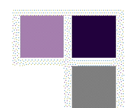
Here the data stored within the Data Structure is
12,15,29,45.
As you can see, the pointer with 12 points to the next linked list which is 15
which points to 29 and so on.
This is just a conceptual idea. In Reality all this data is stored in random
places in memory. Using Pointers help us to get all the data in order.
While Adding Data to a Linked List we check for previously added Linked Lists.
Then we reach the last node of the List where the pointer value is NULL and
point it to our newly created linked list, else if there is no previously
existing linked list we simply add one and set it's pointer to NULL.
Deletion is more complex. Suppose we want to delete the data 15. Then we first
find 15. Then we point the pointer which is present with 12 to the data in 29.
Then we delete the node which contains 15 as it's data.
Studying the Following Source code will help you

understand and Appreciate the

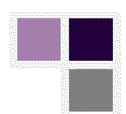
Linked List:

cpp

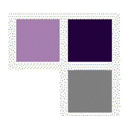
```
#include <iostream>
 using namespace std;
 class linklist
{
     private:
      struct node
    {
     int data;
         node *link;
       }*p;
  public:
     linklist();
       void append( int num );
       void add_as_first( int num );
        void addafter( int c, int num );
```



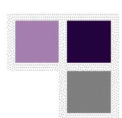
```
        void del( int num );
      void display();
      int count();
       ~linklist();
   };
linklist::linklist()
{
   p=NULL;
  }
void linklist::append(int num)
{   node *q,*t;
  if( p == NULL )
  {
    p = new node;
   p->data = num;
  p->link = NULL;
 }
  else
```



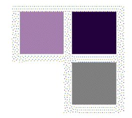
```
 {
   q = p;
    while( q->link != NULL )
   q = q->link;
  t = new node;
  t ->data = num;
 t->link = NULL;
 q->link = t;
}
 }
void linklist::add_as_first(int num)
{
node *q;
q = new node;
q ->data = num;
 q ->link = p;
p = q;
}
```



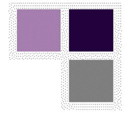
```
void linklist::addafter( int c, int num)
 {
 node *q,*t;   int i;
 for(i=0,q=p;i<c;i++)
  {
 q = q->link;
   if( q == NULL )
  {
 cout<<"\nThere are less than "<<c<<" elements.";
  return;
   }
   }
t = new node;
 t->data = num;
 t->link = q->link;
 q->link = t;
}
void linklist::del( int num )
```



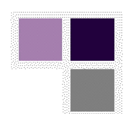
```
{
 node *q,*r;    q = p;
  if( q->data == num )
  {
  p = q->link;
  delete q;
  return;
  }
 r = q;
 while( q!=NULL )
 {
  if( q->data == num )
   {
  r->link = q->link;
     delete q;
   return;
  }
  r = q;
```



```
    q = q->link;
  }
  cout<<"\nElement "<<num<<" not Found.";
}
void linklist::display()
{
 node *q;
   cout<<endl;
  for( q = p ; q != NULL ; q = q->link )
  cout<<endl<<q->data;
}
int linklist::count()
{
 node *q;
 int c=0;
  for( q=p ; q != NULL ; q = q->link )
    c++;    return c;
}
```



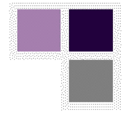
```
linklist::~linklist()
{
 node *q;
if( p == NULL )
  return;
  while( p != NULL )
 {
  q = p->link;
   delete p;
    p = q;
 }
 }
int main()
  {   linklist ll;
  cout<<"No. of elements = "<<ll.count();
  ll.append(12);
  ll.append(13);
  ll.append(23);
```



```
 ll.append(43);
 ll.append(44);
 ll.append(50);
 ll.add_as_first(2);
 ll.add_as_first(1);
ll.addafter(3,333);
ll.addafter(6,666);
 ll.display();
 cout<<"\nNo. of elements = "<<ll.count();
ll.del(333);
ll.del(12);
 ll.del(98);
 cout<<"\nNo. of elements = "<<ll.count();
 return 0;
}
```

OUTPUT:

No. of elements = 0



1

2

12

13

333

23

43

666

44

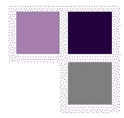
50

No. of elements = 10

Element 98 not Found.

No. of elements = 8

Here as you see, the class contains a structure node that consists of an integer
type for data and a pointer pointing to another node structure. Here we maintain
a node pointer p that always points to the first item in the



list. Here is a

list of the functions that are used in the data structure.

cpp

```
linklist();                     // CONSTRUCTOR

 void append( int num );          // ADD AT END OF LIST

void add_as_first( int num );     // ADD TO BEGINNING OF LIST

 void addafter( int c, int num );  // ADD DATA num AFTER POSTION
c

 void del( int num );             // DELETE DATA num

 void display();                  // DISPLAY LINKED LIST

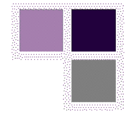
 int count();                     // NUMBER OF ITEMS IN LIST

~linklist();                      // DESTRUCTOR
```

Many places you will see statements like q=q->link inside a loop. This statement

just shifts the pointer from one node to the other. the Destructor as well as

the del() function use the delete operator to deallocate space that was
previously allocated by the new operator. The Rest should be clear if you have a
basic understanding of pointers.
The advantage of using pointers is that you dont have to worry about wasting
space by allocating a lot of memory beforehand. As the need for data increases,
memory is allocated accordingly. But the flip side is that to access each node
we have to iterate through each node till we reach the desired node. That's why
linked lists have different forms of themselves for easier access. For example
Circular and Doubly Linked Lists. Circular Linked Lists are those in which the
last node always points to the first node in the list. Doubly Linked Lists
contain two pointers, one that points to the next node and the other that points
to the previous node.
I shall only give source code for Circular Linked List, while code for doubly



linked lists is given as an excercise. However, if you cant write it...you are
free to contact me at born2c0de AT dreamincode DOT net

VI. STACKS USING LINKED LISTS

Here, we use the same concept of the stack but eliminate the MAXIMUM data items
constraint. Since we shall be using Linked Lists to store data in the stack,
the Stack can hold as much as data as it wants as long as the data is within
Memory Limits.Here's the code:

cpp

```
#include <iostream>

 using namespace std;

 struct node {    int data;    node *link; };

 class lstack

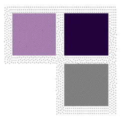
{

      private:

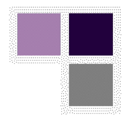
          node* top;

     public:
```

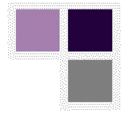
```
lstack()
{
  top=NULL;
}
void push(int n)
{
node *tmp;
tmp=new node;
if(tmp==NULL)
  cout<<"\nSTACK FULL";
tmp->data=n;
tmp->link=top;
top=tmp;
}
int pop()
{
if(top==NULL)
{
```



```
        cout<<"\nSTACK EMPTY";
       return NULL;
    }
    node *tmp;
     int n;
      tmp=top;
     n=tmp->data;
      top=top->link;
     delete tmp;
    return n;
    }
  ~lstack()
  {
   if(top==NULL)
      return;
     node *tmp;
    while(top!=NULL)
     {
```



```
tmp=top;
 top=top->link;
   delete tmp;
}
}
};
int main()
{
lstack s;
 s.push(11);
s.push(101);
 s.push(99);
s.push(78);
 cout<<"Item Popped = "<<s.pop()<<endl;
cout<<"Item Popped = "<<s.pop()<<endl;
 cout<<"Item Popped = "<<s.pop()<<endl;
 return 0;
}
```

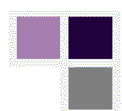


VII. QUEUES USING LINKED LISTS

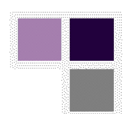
Similar to the one above, the queued linked list removes the maximum data limit
as well. Here is the code:

cpp

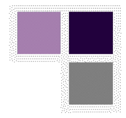
```
#include <iostream>
 using namespace std;
 struct node
   {
   int data;
   node *link;
 };
 class lqueue
{
  private:
   node *front,*rear;
 public:
```



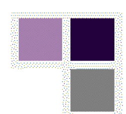
```
lqueue()
 {
  front=NULL;
 rear=NULL;
 }
void add(int n)
 {
  node *tmp;
  tmp=new node;
  if(tmp==NULL)
   cout<<"\nQUEUE FULL";
  tmp->data=n;
 tmp->link=NULL;
  if(front==NULL)
 {
  rear=front=tmp;
     return;
}
```



```
rear->link=tmp;
        rear=rear->link;
 }
int del()
{
if(front==NULL)
 {
        cout<<"\nQUEUE EMPTY";
      return NULL;
  }
   node *tmp;
    int n;
    n=front->data;
   tmp=front;
    front=front->link;
     delete tmp;
   return n;
 }
```



```
~lqueue()
 {
if(front==NULL)
 return;
node *tmp;
 while(front!=NULL)
 {
    tmp=front;
     front=front->link;
     delete tmp;
 }
}
   };
int main()
      {
  lqueue q;
  q.add(11);
q.add(22);
```



```
q.add(33);
q.add(44);
q.add(55);
cout<<"\nItem Deleted = "<<q.del();
cout<<"\nItem Deleted = "<<q.del();
cout<<"\nItem Deleted = "<<q.del();
return 0;
}
```

VIII. CIRCULAR QUEUES

Circular Linked Lists are just like normal linked lists except that the pointer

of the last item in the list points to the first item in the

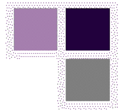
list. You must be
wondering...Why would anyone ever want to do such a thing? Well...did you know
that circular linked lists are used almost in every situation, they are infact
used in Electronic Advertisements where each ad is added to the list and is
displayed. After the last ad is displayed the linked list will automatically
display the first ad in the List.

Now let us see how we can implement the Circular Linked List. I've written this
code in much more detail plus I've included a SLIDESHOW Feature that shows the
Data in the List after a time-period is elapsed. It goes on displaying the data
until a key is pressed. Have a look:

Note:

If you aren't using Windows, follow these steps:

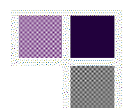
1. Remove #include <windows.h>
2. Remove #include <conio.h>
3. Remove the slideshow() and wait() function from CL_list class.



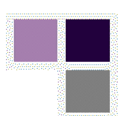
4. Remove the slideshow() function call in main().

cpp

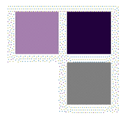
```
#include <windows.h>
 #include <iostream>
 #include <conio.h>
 using namespace std;
  class CL_list
      {
   private:
     struct node
      {
      int data;
      node *link;
     };
    struct node *p;
   public:
    CL_list();
    CL_list(CL_list& l);
```



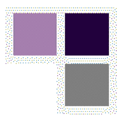
```
   ~CL_list();
  void add(int);
  void del();
 void addatbeg(int);
  void display();
  void slideshow(float,int,int);
   int count();
  void wait(float);
  bool operator ==(CL_list);
   bool operator !=(CL_list);
  void operator =(CL_list); };
 CL_list::CL_list()
 {
p=NULL;
   }
 CL_list::CL_list(CL_list& l)
 {
   node *x;
```



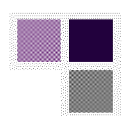
```
p=NULL;
x=l.p;
if(x==NULL)
   return;
     for(int i=1;i<=l.count();i++)
   {
    add(x->data);
   x=x->link;
   }
      }
     CL_list::~CL_list()
      {
       node *q,*t;
  q=p;
   t=p;
    if(p==NULL)
       return;
      while(q->link!=t)
```



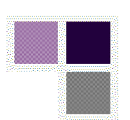
```
{
  p=q;
  q=q->link;
  delete p;
}
  p=q;
  delete p;
}
void CL_list::add(int n)
{
   if(p==NULL)
   {
     node *q;
  q=new node;
     q->data=n;
   q->link=q;
     p=q;
 return;
```



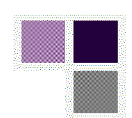
```
}
 node *q;
q=p;
  while(q->link != p)
q=q->link;
 node *t;
 t=new node;
 t->data=n;
    t->link=p;
 q->link=t;
 }
 void CL_list::display()
  {
if(p==NULL)
 {
    cout<<"EMPTY LIST\n";
    return;
}
```



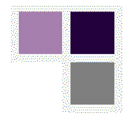
```
node *q;
q=p;
for(int i=1;i<=this->count();i++)
{
cout<<q->data<<endl;
q=q->link;
}
}
int CL_list::count()
{
node *q;
q=p;
int c=0;
if(p==NULL)
return 0;
else
c++;
while(q->link != p)
```



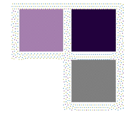
```
{
c++;
q=q->link;
}
return c;
}
void CL_list::del()
{
if(p==NULL)
return;
if(p->link==p)
{
p=NULL;
}
else
{
node *q;
q=p;
```



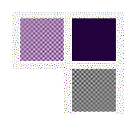
```
      while(q->link != p )
        q=q->link;
    q->link=p->link;
    q=p;
    p=(q->link == NULL ? NULL : p->link);
    delete q;
     }
  }
 void CL_list::addatbeg(int n)
 {
 node *q,*t;
   q=p;
   while(q->link!=p)
     q=q->link;
   t=new node;
    t->data=n;
   t->link=p;
q->link=t;
```



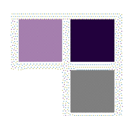
```
  p=t;
 }
void CL_list::slideshow(float dlay,int x,int y)
{
/*  if(p==NULL)
  {
   gotoxy(x,y);
     cout<<"EMPTY LIST\n";
     return;
   }
node *q;
 q=p;
 while(!kbhit())
    {
    gotoxy(x,y);
    cout<<"               ";
    gotoxy(x,y);
   cout<<q->data;
```



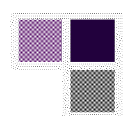
```
    wait(dlay);
q=q->link;      }*/
 }
 void CL_list::wait(float t)
  {
 long time=GetTickCount()+(t*1000L);
   while(GetTickCount()<=time)
   {
  /*       WAIT !!! */
 }
 }
 bool CL_list::operator ==(CL_list t)
{      if(t.p==NULL && p==NULL)
     return 1;
  if(this->count() != t.count())
      return 0;
  node *q;
   q=p;
```



```
 bool flag;
 flag=1;
node *a;
a=t.p;
 for(int i=1;i<=count();i++)
{
if(a->data!=q->data)
     flag=0;
    a=a->link;
    q=q->link;
 }
 if(a->data!=q->data)
    flag=0;
  return flag;
}
bool CL_list::operator !=(CL_list t)
 {
 return !(this->operator==(t));
```

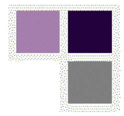


```
 }
int main()
{
 CL_list a;
 a.add(1);
  a.add(2);
 a.add(3);
 a.add(4);
 a.addatbeg(128);
 a.del(); // 128 is deleted
 cout<<"\nLIST DATA:\n";
  a.display();
 CL_list b=a;
  if(b!=a)
   cout<<endl<<"NOT EQUAL"<<endl;
  else
  cout<<endl<<"EQUAL"<<endl;
a.slideshow(1,13,13);
```



```
  return 0;
}
```

Here once again we have made sure that the last node always points to the first
node. Everything else seems fine. Comments should be enough to explain the code.
The interesting part of this code is the slideshow() function. It plainly
displays the list in an infinite loop which can be terminated by pressing a key.



The wait() function allows the delay while the kbhit() function checks for a
keypress.

Now comes the test. Write a similar linked list only with the following changes:

1) Structure node should be like this:

cpp

```
struct node
  {
  int data;
   node *next;  // Pointer to Next Node
   node *prev;  // Pointer to Previous Node
  };
```

2) Remember that while adding and deleting the next and previous pointers have
to be set up accordingly.

3) Include a display function with a parameter like this:

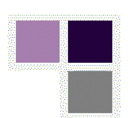
cpp

```
void linklist::display(int type)
  {
       if(type==1)
     {
    // Code for output from First Node to Last node
       }
     else
      {
    // Code for output from Last Node to First
      }
  }
```

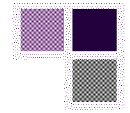
This function is really easy to write if you understand how to use both the next
and previous pointers.

If you still cant write the code mail me with your difficulties at my email add:
born2c0de AT dreamincode DOT net

IX. BINARY SEARCH TREES



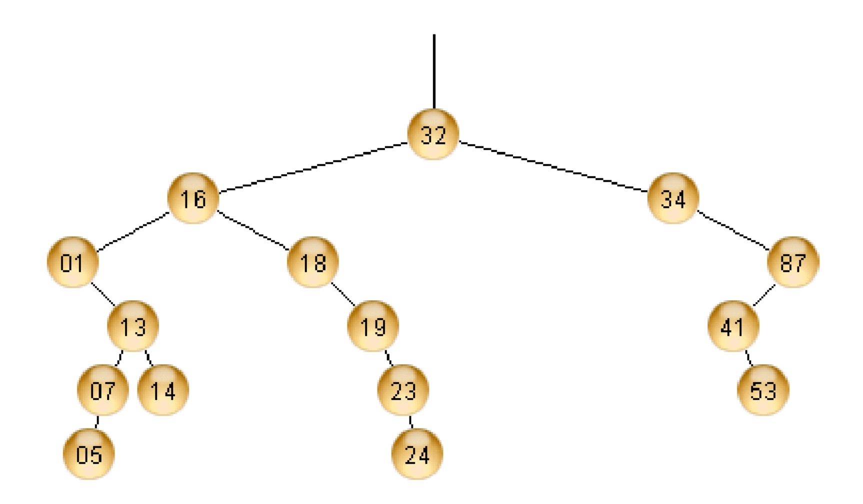
Uptil now all data structures that we have covered (Stack,Queue,Linked List) are
linear in nature ie. they have a definate order of placement. Now we shall study
Binary Trees which requires a different thought process as it is a non linear
data structure.
A Binary Tree consists of a main node known as the Root. The Root then has two
sub-sections, ie. the left and the right half. The data subsequently stored
after the root is created depends on it's value compared to the root.
Suppose the root value is 10 and the Value to be added is 15, then the data is
added to the right section of the root.
The Basic idea is that every node can be thought of a binary tree itself. Each
node has two pointers, one to the left and the other to the right. Depending on
the value to be stored, it is placed after a node's right pointer if the value
of the node is lesser than the one to be added or the node's left pointer if



viceversa.

Let's take an Example. To add the Following List of Numbers, we end up with a Binary Tree like this:

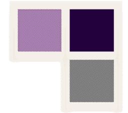
32 16 34 1 87 13 7 18 14 19 23 24 41 5 53



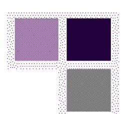
Here's How:

**: KEEP ADDING DATA IN THE TREE ON PAPER AFTER EACH STEP BELOW TO UNDERSTAND HOW THE TREE IS FORMED.

1. Since **32** is the First Number to be added, 32 becomes the root of the tree.
2. Next Number is **16** which is lesser than 32 Hence 16 becomes left node of 32.



3. **34**. Since 34 > 32 , 34 becomes the right node of the ROOT.
4. **1**. Since 1 < 32 we jump to the left node of the ROOT. But since the left node
   has already been taken we test 1 once again. Since 1 < 16, 1 becomes the left
   node of 16.
5. **87**. Since 87 > 32 we jump to the right node of the root. Once again this
   space is occupied by 34. Now since 87 > 34, 87 becomes the right node of 34.
6. **13**. Since 13 < 32 we jump to left node of the root. There, 13 < 16 so we
   continue towards the left node of 16. There 13 > 1, so 13 becomes the right
   node of 1.
7. Similarly work out addition till the end ie. before Number **53**.
8. **53**. Since 53 > 32 we jump to the right node of the root. There 53 > 34 so we
   continue to the right node of 34. There 53 < 87 so we continue towards the
   left node of 87. There 53 > 41 so we jump to the right node of 41. Since the



Right node of 41 is empty 53 becomes the right node of 41.

This should give you an idea of how a Binary Tree works. You must know that:

1. The linking of nodes to nodes in a Binary Tree is one to one in nature
   ie. a node cannot be pointed by more than 1 node.
2. A Node can point to two different sub-nodes at the most.

Here in the binary tree above there are a few nodes whose left and right
pointers are empty ie. they have no sub-node attached to them. So Nodes 5,14,18,
19,23,24,41 have their left nodes empty.

There are three popular ways to display a Binary Tree. Displaying the trees
contents is known as transversal. There are three ways of transversing a tree iw.
in inorder,preorder and postorder transversal methods. Description of each is
shown below:

PREORDER:

1. Visit the root.
2. Transverse the left leaf in preorder.
3. Transverse the right leaf in preorder.

INORDER:

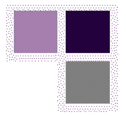
1. Transverse the left leaf in inorder.
2. Visit the root.
3. Transverse the right leaf in inorder.

POSTORDER:

1. Transverse the left leaf in postorder.
2. Transverse the right leaf in postorder.
3. Visit the root.

Writing code for these three methods are simple if we understand the recursive
nature of a binary tree. Binary tree is recursive, as in each node can be
thought of a binary tree itself. It's just the order of displaying data that
makes a difference for transversal.

Deletion from a Binary Tree is a bit more difficult to



understand. For now just

remember that for deleting a node, it is replaced with it's next inorder

successor. I'll explain everything after the Binary Tree code.

Now that you've got all your Binary Tree Fundas clear, let's move on with the

Source code.

cpp

```
#include <iostream>

 using namespace std;

 #define YES 1 #define NO 0

 class tree

 {

  private:

     struct leaf

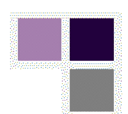
         {

          int data;

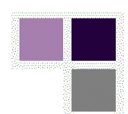
              leaf *l;

              leaf *r;
```

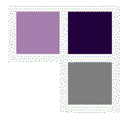
```
    };
     struct leaf *p;
   public:
    tree();
      ~tree();
      void destruct(leaf *q);
      tree(tree& a);
       void findparent(int n,int &found,leaf* &parent);
    void findfordel(int n,int &found,leaf *&parent,leaf* &x);
        void add(int n);
       void transverse();
     void in(leaf *q);
  void pre(leaf *q);
  void post(leaf *q);
     void del(int n);
};
    tree::tree()
{
```



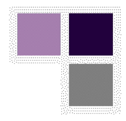
```
 p=NULL; } tree::~tree()
{
  destruct(p);
}
void tree::destruct(leaf *q)
{
   if(q!=NULL
{
   destruct(q->l);
       del(q->data);
       destruct(q->r);
   }
}
void tree::findparent(int n,int &found,leaf *&parent)
{
   leaf *q;
    found=NO;
   parent=NULL;
```



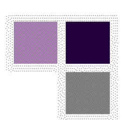
```
if(p==NULL)
  return;
q=p;
while(q!=NULL)
 {
 if(q->data==n)
  {
     found=YES;
     return;
  }
   if(q->data>n)
  {
     parent=q;
     q=q->l;
 }
 else
   {
     parent=q;
```



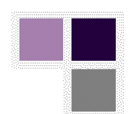
```
            q=q->r;
      }
}
}
void tree::add(int n)
{
 int found;
   leaf *t,*parent;
   findparent(n,found,parent);
  if(found==YES)
      cout<<"\nSuch a Node Exists";
    else
    {
     t=new leaf;
       t->data=n;
        t->l=NULL;
      t->r=NULL;
     if(parent==NULL)
```



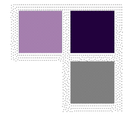
```
          p=t;
       else
         parent->data > n ? parent->l=t : parent->r=t;
   }
  }
void tree::transverse()
 {
 int c;
  cout<<"\n1.InOrder\n2.Preorder\n3.Postorder\nChoice: ";
    cin>>c;
    switch©
    {
       case 1:
            in(p);
             break;
      case 2:
         pre(p);
            break;
```



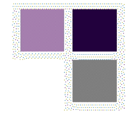
```
        case 3:
         post(p);
          break;
   }
  }
void tree::in(leaf *q)
{     if(q!=NULL)
   {
     in(q->l);
       cout<<"\t"<<q->data<<endl;
        in(q->r);
   }
  }
void tree::pre(leaf *q)
{     if(q!=NULL
{
     cout<<"\t"<<q->data<<endl;
        pre(q->l);
```



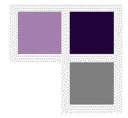
```
      pre(q->r);
  }
   }
void tree::post(leaf *q)
{
  if(q!=NULL)
   {
 post(q->l);
     post(q->r);
         cout<<"\t"<<q->data<<endl;
   }
   }
void tree::findfordel(int n,int &found,leaf *&parent,leaf *&x)
{
leaf *q
found=0;
  parent=NULL;
 if(p==NULL)
```



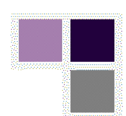
```
   return;
 q=p;
  while(q!=NULL)
   {
   if(q->data==n)
       {
        found=1;
      x=q;
        return;
    }
   if(q->data>n)
    {
     parent=q;
         q=q->l;
    }
   else
    {
     parent=q;
```



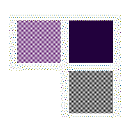
```
        q=q->r;
      }
    }
  }
void tree::del(int num)
 {
    leaf *parent,*x,*xsucc;
   int found;      // If EMPTY TREE
  if(p==NULL)
   {
   cout<<"\nTree is Empty";
      return;
    }
  parent=x=NULL;
    findfordel(num,found,parent,x);
     if(found==0)
     {
     cout<<"\nNode to be deleted NOT FOUND";
```



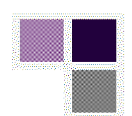
```
    return;
}
// If the node to be deleted has 2 leaves
  if(x->l != NULL && x->r != NULL)
   {
     parent=x;
    xsucc=x->r;
     while(xsucc->l != NULL)
     {
     parent=xsucc;
      xsucc=xsucc->l;
     }
    x->data=xsucc->data;
     x=xsucc;
   }
 // if the node to be deleted has no child
 if(x->l == NULL && x->r == NULL)
 {          if(parent->r == x)
```



```
        parent->r=NULL;
      else
         parent->l=NULL;
        delete x;
       return;
     }
    // if node has only right leaf
   if(x->l == NULL && x->r != NULL )
    {
      if(parent->l == x)
        parent->l=x->r;
      else
      parent->r=x->r;
        delete x;
      return;
 }
   // if node to be deleted has only left child
    if(x->l != NULL && x->r == NULL)
```



```
    {
    if(parent->l == x)
        parent->l=x->l;
      else
    parent->r=x->l;
      delete x;
     return;
  }
}
int main()
{
  tree t;
  int data[]={32,16,34,1,87,13,7,18,14,19,23,24,41,5,53};
   for(int iter=0 ; iter < 15 ; i++)
      t.add(data[iter]);
    t.transverse();
  t.del(16);
    t.transverse();
```



```
t.del(41);
t.transverse();
return 0;
}
```

OUTPUT:

1.InOrder

2.Preorder

3.Postorder

Choice: 1

1

5

7

13

14

16

18

19

23

24

32

34

41

53

87

1.InOrder

2.Preorder

3.Postorder

Choice: 2

32

18

1

13

7

5

14

19

23

24

34

87

41

53

1.InOrder

2.Preorder

3.Postorder

Choice: 3

5

7

14

13

1

24

23

19

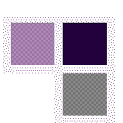
18

53

87

34

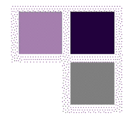
32



Press any key to continue

NOTE: Visual C++ may give Runtime Errors with this code. Compile with Turbo C++.

Just by looking at the output you might realise that we can print out the whole
tree in ascending order by using inorder transversal. Infact Binary Trees are
used for Searching [Binary Search Trees {BST}] as well as in Sorting.



The Addition of data part seems fine. Only the deletion bit needs to be
explained.

For deletion of data there are a few cases to be considered:

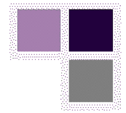
1. If the leaf to be deleted is not found.
2. If the leaf to be deleted has no sub-leafs.
3. If the leaf to be deleted has 1 sub-leaf.
4. If the leaf to be deleted has 2 sub-leafs.

CASE 1:

Dealing with this case is simple, we simply display an error message.

CASE 2:

Since the node has no sub-nodes, the memory occupied by this
should be freed and either the left link or the right link of the parent of this
node should be set to NULL. Which of these should be set to NULL depends upon
whether the node being deleted is a left child or a right child of its parent.



CASE 3:

In the third case we just adjust the pointer of the parent of the leaf to be
deleted such that after deletion it points to the child of the node being
deleted.

CASE 4:

The last case in which the leaf to be deleted has to sub-leaves of its own is
rather complicated.The whole logic is to locate the inorder successor, copy it's
data and reduce the problem to simple deletion of a node with one or zero leaves.
Consider in the above program...(Refer to the previous tree as well) when we are
deleting 16 we search for the next inorder successor. So we simply set the data
value to 5 and delete the node with value 5 as shown for cases 2 and 3.

That's It! **phew**

Binary Trees are used for various other things which even

include

Compression algorithms,binary searching,sorting etc. A lot of Huffman,

Shannon-Fano and other Compression algorithms use Binary Trees. If you want

source code of these Compression codes you can freely contact me at my email.

X. CONTACT ME

That wraps up this Data Structure Tutorial. There are plenty of other data

structures which I'd love to mention such as Sparse Matrices, Graphs, B-Trees,

B+ Trees, AVL Trees etc. but since the aim of this tutorial was to give an

introduction to Data Structures I decided not to include them in this text.

Maybe I can save them for another Tutorial which I may write in the future.

If you have any problems in understanding the text or the source code, please

post in the forums instead of mailing me.

However, if you have any comments, suggestions or error reports, feel free to

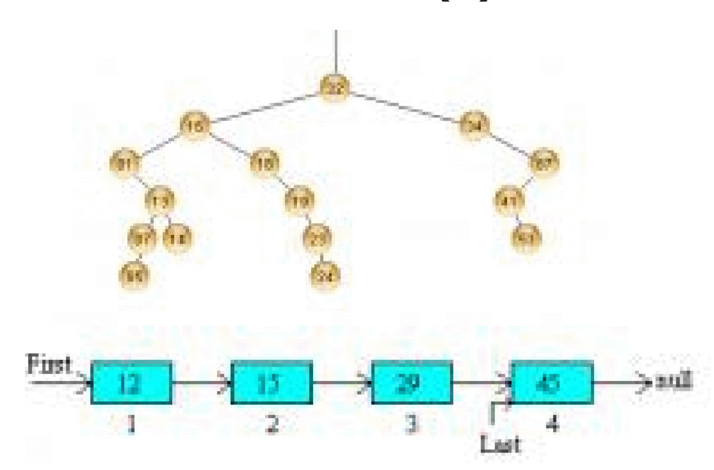
contact me at: **born2c0de AT dreamincode DOT net**

***Update** :* I'll be submitting a sequel to this tutorial that discusses many more data structures very soon. Keep checking for updates.

For the downloadable text version, download the text file below.

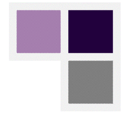
datastruct.txt (36.81k) Number of downloads: 5168

Attached thumbnail(s)





1. Theory And Problems of Data Structures. Seymour Lipschtz .1986 .



2. Stubbs , Danil . Data Structure with Abstract Data Types , 1989

3. Hale , Guy J. , Applied Data Structures , 1987.

4. Lipschutz , Seymour ,Outline of Theory and Problems of Data Structures , 1986